Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 1: Preface

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Preface ..... **iv**

Organization of the PRM iv

Device Tags and Definitions vi

Reserved Bits and Software Compatibility vi

Terminology vii

Preface

This section includes descriptions of the following:

- Organization of the PRM
- Devices and Device Tag Definitions
- Reserved Bits and Software Compatibility
- Terminology

Organization of the PRM

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

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

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

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

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

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

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

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

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

Topic
Media VDBOX covers the CHV **Multi-Format Codec (MFX) Engine**, the hardware fixed-function pipeline that includes both multi-format decoding (MFD) and multi-format encoding (MFC).
Media VEBOX discusses the Video Enhancement pipeline. Topics include denoise filtering, deinterlacing, color processing, auto contrast enhancement, the capture pipe, output statistics, and video enhancement engine functions.
HEVC covers the High Efficiency Video Coding (HEVC) Codec Pipeline (HCP), which is a fixed function hardware video codec pipeline responsible for encoding and decoding HEVC streams, with a target resolution of 4kx2k at 60 frames per second.

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

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

Topic
Memory-Mapped I/O (MMIO) lists Slice Registers and Die Recovery Units including Start and End Range offsets.
GTI describes the **GT Interface (GTI)**, which consists of a mid-level cache, super queue, IDI and CRAB controllers, and GT power management controls. This section provides information about request types and flows, fence operations, power management, and error handling. It also lists the GTI registers that manage these and other functions.

Device Tags and Definitions

The following table lists device "tags" (abbreviations, projects) used in various parts of this document as aliases for the device names.

Information without any device tagging is applicable to all CHV devices.

Project	Program Name	Graphics Architecture	SKU	Product Name / Description
CHV BSW	CherryView Braswell	Gen8LP		Low-power variation of Broadwell, which was a die shrink of Haswell from 22 nm to 14 nm.

Reserved Bits and Software Compatibility

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

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

Terminology

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

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

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

Term	Abbr.	Definition
Immediate integer vector	V	A numerical data type of 32 bits, an immediate integer vector of type V contains 8 signed integer elements with 4 bits each. The 4-bit integer element is in 2's complement form. It may be used to specify the type of an immediate operand in an instruction.
Index Buffer	IB	Buffer in memory containing vertex indices.
Intel Architecture	IA	
Instance	--	In the context of the VF unit, an instance is one of a sequence of sets of similar primitive data. Each set has identical vertex data but may have unique instance data that differentiates it from other sets in the sequence.
Instruction	--	Data in memory directing an EU operation. Instructions are fetched from memory, stored in a cache, and executed on one or more cores. Not to be confused with commands which are fetched and parsed by the command streamer and dispatched down the 3D or Media pipeline.
Instruction Pointer	IP	The address (really an offset) of the instruction currently being fetched by an EU. Each EU has its own IP.
Instruction Set Architecture	ISA	The ISA describes the instructions supported by an EU.
Instruction State Cache	ISC	On-chip memory that holds recently-used instructions and state variable values.
Interface Descriptor	--	Media analog of a State Descriptor.
Intermediate Z	IZ	Completion of the Z (depth) test at the front end of the Windower/Masker unit when certain conditions are met (no alpha, no pixel-shader computed Z values, etc.).
Inverse Discrete Cosine Transform	IDCT	The stage in the video decoding pipe between IQ and MC.
Inverse Quantization	IQ	A stage in the video decoding pipe between IS and IDCT.
JIT	--	Just-in-time compiler, aka "jitter".
Kernel	--	A sequence of instructions that is logically part of the driver or generated by the jitter. Differentiated from a Shader which is an application supplied program that is translated by the jitter to instructions.
Logical Ring Context Area	LRCA	Memory area used to store contents of registers and state information required for initiating and resuming communication between software application and hardware graphics pipeline via Ring Buffers.
Least Significant Bit	LSB	The bit with the lowest bit position within a group of bits, which could be a bit group, DWord, field, instruction, memory range, register, or structure. For example, bit 0 of a DWord.
MathBox	--	See *Extended Math Unit*
Media	--	Term for operations such as video decode and encode that are normally performed by the Media pipeline.
Media Pipeline	--	Fixed function stages dedicated to media and "generic" processing, sometimes referred to as the generic pipeline.

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

Term	Abbr.	Definition
Quad Word (QWord)	QW	A fundamental data type, QW represents 8 bytes.
Rasterization	--	Conversion of an object represented by vertices into the set of pixels that make up the object.
Region-based addressing	--	Collective term for the register addressing modes available in the EU instruction set that permit discontiguous register data to be fetched and used as a single operand.
Render Cache	RC	Cache in which pixel color and depth information is written before being written to memory, and where prior pixel destination attributes are read in preparation for blending and Z test.
Render Target	RT	A destination surface in memory where render results are written.
Render Target Array Index	--	Selector of which of several render targets the current operation is targeting.
Resource Streamer	RS	Functional unit of the Graphics Processing Engine that examines the commands in the ring buffer in an attempt to pre-process certain long latency items for the remainder of the graphics processing.
Root Thread	--	A root-node thread. A thread corresponds to a root-node in a thread generation hierarchy. It is a kind of thread associated with the media fixed function pipeline. A root thread is originated from the VFE unit and forwarded to the Thread Dispatcher by the TS unit. A root thread may or may not have child threads. A root thread may have scratch memory managed by TS. A root thread with children has its URB resource managed by the VFE.
Sampler	--	Shared function that samples textures and reads data from buffers on behalf of EU programs.
Scratch Space	--	Memory allocated to the subsystem that is used by EU threads for data storage that exceeds their register allocation, persistent storage, storage of mask stack entries beyond the first 16, etc.
Shader	--	A program supplied by the application in a high level shader language, and translated to instructions by the jitter.
Shared Function	SF	Function unit that is shared by EUs. EUs send messages to shared functions, that consume the data and may return results. The Sampler, Data Port, and Extended Math unit are all shared functions.
Shared Function ID	SFID	Unique identifier used by kernels and shaders to target shared functions and to identify their returned messages.
Single Instruction Multiple Data	SIMD	A parallel processing architecture that exploits data parallelism at the instruction level. It can also be used to describe the instructions in such an architecture or to describe the amount of data parallelism in a particular instruction (SIMD8 for example).
Source	--	Describes an input or read operand.
Spawn	--	To initiate a thread for execution on an EU. Done by the thread spawner as well as most FF units in the 3D Pipeline.

Term	Abbr.	Definition
Sprite Point	--	Point object using full range texture coordinates. Points that are not sprite points use the texture coordinates of the point's center across the entire point object.
State Descriptor	--	Blocks in memory that describe the state associated with a particular FF, including its associated kernel pointer, kernel resource allowances, and a pointer to its surface state.
State Register	SR	The read-only registers containing the state information of the current thread, including the EUID/TID, Dispatcher Mask, and System IP.
State Variable	SV	An individual state element that can be varied to change the way given primitives are rendered or media objects processed. On state variables persist only in memory and are cached as needed by rendering/processing operations except for a small amount of non-pipelined state.
Stream Output	--	A term for writing the output of a FF unit directly to a memory buffer instead of, or in addition to, the output passing to the next FF unit in the pipeline. Currently only supported for the Geometry Shader (GS) FF unit.
Strips and Fans	SF	Fixed function unit whose main function is to decompose primitive topologies such as strips and fans into primitives or objects.
Sub-Register	--	Subfield of a SIMD register. A SIMD register is an aligned fixed size register for a register file or a register type. For example, a GRF register, *r2*, is a 256-bits wide, 256-bit aligned register. A sub-register, *r2.3:d*, is the fourth dword of GRF register *r2*.
Subsystem	--	The name given to the resources shared by the FF units, including shared functions and EUs.
Surface	--	A rendering operand or destination, including textures, buffers, and render targets.
Surface State Base Pointer	--	Base address used when referencing binding table and surface state data.
Synchronized Root Thread	--	A root thread that is dispatched by TS upon a 'dispatch root thread' message.
System IP	SIP	There is one global System IP register for all the threads. From a thread's point of view, this is a virtual read only register. Upon an exception, hardware performs some bookkeeping and then jumps to SIP.
System Routine	--	Sequence of instructions that handles exceptions. SIP is programmed to point to this routine, and all threads encountering an exception will call it.
Thread	--	An instance of a kernel program executed on an EU. The life cycle for a thread starts from the executing the first instruction after being dispatched from Thread Dispatcher to an EU to the execution of the last instruction – a *send* instruction with EOT that signals the thread termination. Threads in the system may be independent from each other or communicate with each other through Message Gateway share function.
Thread Dispatcher	TD	Functional unit that arbitrates thread initiation requests from Fixed Functions units and instantiates the threads on EUs.

Term	Abbr.	Definition
Thread Identifier	TID	The field within a thread state register (SR0) that identifies which thread slots on an EU a thread occupies. A thread can be uniquely identified by the EUID and TID.
Thread Payload	--	Before a thread starting execution, some amount of data is pre-loaded into the thread's GRF (starting at r0). This data is typically a combination of control information provided by the spawning entity (FF Unit) and data read from the URB.
Thread Spawner	TS	The second and the last fixed function stage of the media pipeline that initiates new threads on behalf of generic/media processing.
Topology	--	See *Primitive Topology*.
Unified Return Buffer	URB	The on-chip memory managed/shared by Fixed Functions in order for a thread to return data that will be consumed either by a Fixed Function or other threads.
Unsigned Byte integer	UB	A numerical data type of 8 bits.
Unsigned Double Word integer	UD	A numerical data type of 32 bits. It may be used to specify the type of an operand in an instruction.
Unsigned Word integer	UW	A numerical data type of 16 bits. It may be used to specify the type of an operand in an instruction.
Unsynchronized Root Thread	--	A root thread that is automatically dispatched by TS.
URB Dereference	--	
URB Entry	UE	URB Entry: A logical entity stored in the URB (such as a vertex), referenced via a URB Handle.
URB Entry Allocation Size	--	Number of URB entries allocated to a Fixed Function unit.
URB Fence	Fence	Virtual, movable boundaries between the URB regions owned by each FF unit.
URB Handle	--	A unique identifier for an URB entry that is passed down a pipeline.
URB Reference	--	
Variable Length Decode	VLD	The first stage of the video decoding pipe that consists mainly of bit-wide operations.
Vertex Buffer	VB	Buffer in memory containing vertex attributes.
Vertex Cache	VC	Cache of Vertex URB Entry (VUE) handles tagged with vertex indices. See the VS chapter for details on this cache.
Vertex Fetcher	VF	The first FF unit in the 3D Pipeline responsible for fetching vertex data from memory. Sometimes referred to as the Vertex Formatter.
Vertex Header	--	Vertex data required for every vertex appearing at the beginning of a Vertex URB Entry.
Vertex ID	--	Unique ID for each vertex that can optionally be included in vertex attribute data sent down the pipeline and used by kernel/shader threads.

Term	Abbr.	Definition
Vertex Shader	VS	An API-supplied program that calculates vertex attributes. Also refers to the FF unit that dispatches threads to “shade” (calculate attributes for) vertices.
Vertex URB Entry	VUE	An URB entry that contains data for a specific vertex.
Vertical Stride	VertStride	The distance in element-sized units between 2 vertically-adjacent elements of a region-based GRF access.
Video Front End	VFE	The first fixed function in the generic pipeline; performs fixed-function media operations.
Viewport	VP	
Windower IZ	WIZ	Term for Windower/Masker that encapsulates its early (“intermediate”) depth test function.
Windower/Masker	WM	Fixed function triangle/line rasterizer.
Word	W	A numerical data type of 16 bits, W represents a signed word integer.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

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

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

3DPRIMITIVE 1
3DSTATE_AA_LINE_PARAMETERS 5
3DSTATE_BINDING_TABLE_EDIT_DS 7
3DSTATE_BINDING_TABLE_EDIT_GS 9
3DSTATE_BINDING_TABLE_EDIT_HS 11
3DSTATE_BINDING_TABLE_EDIT_PS 13
3DSTATE_BINDING_TABLE_EDIT_VS 15
3DSTATE_BINDING_TABLE_POINTERS_DS 17
3DSTATE_BINDING_TABLE_POINTERS_GS 19
3DSTATE_BINDING_TABLE_POINTERS_HS 21
3DSTATE_BINDING_TABLE_POINTERS_PS 23
3DSTATE_BINDING_TABLE_POINTERS_VS 25
3DSTATE_BINDING_TABLE_POOL_ALLOC 27
3DSTATE_BLEND_STATE_POINTERS 29
3DSTATE_CC_STATE_POINTERS 31
3DSTATE_CHROMA_KEY 33
3DSTATE_CLEAR_PARAMS 35
3DSTATE_CLIP 37
3DSTATE_CONSTANT_DS 44
3DSTATE_CONSTANT_GS 46
3DSTATE_CONSTANT_HS 48
3DSTATE_CONSTANT_PS 50
3DSTATE_CONSTANT_VS 52
3DSTATE_DEPTH_BUFFER 54
3DSTATE_DRAWING_RECTANGLE 60
3DSTATE_DS 63
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC 72
3DSTATE_DX9_CONSTANTB_PS 74
3DSTATE_DX9_CONSTANTB_VS 76
3DSTATE_DX9_CONSTANTF_PS 78
3DSTATE_DX9_CONSTANTF_VS 80

3DSTATE_DX9_CONSTANTI_PS 82
3DSTATE_DX9_CONSTANTI_VS 84
3DSTATE_DX9_GENERATE_ACTIVE_PS 86
3DSTATE_DX9_GENERATE_ACTIVE_VS 88
3DSTATE_DX9_LOCAL_VALID_PS 90
3DSTATE_DX9_LOCAL_VALID_VS 92
3DSTATE_GATHER_CONSTANT_DS 94
3DSTATE_GATHER_CONSTANT_GS 97
3DSTATE_GATHER_CONSTANT_HS 100
3DSTATE_GATHER_CONSTANT_PS 103
3DSTATE_GATHER_CONSTANT_VS 106
3DSTATE_GATHER_POOL_ALLOC 109
3DSTATE_GS 111
3DSTATE_HIER_DEPTH_BUFFER 122
3DSTATE_HS 125
3DSTATE_INDEX_BUFFER 132
3DSTATE_LINE_STIPPLE 134
3DSTATE_MONOFILTER_SIZE 136
3DSTATE_MULTISAMPLE 138
3DSTATE_POLY_STIPPLE_OFFSET 141
3DSTATE_POLY_STIPPLE_PATTERN 143
3DSTATE_PS_BLEND 144
3DSTATE_PS 146
3DSTATE_PS_EXTRA 154
3DSTATE_PUSH_CONSTANT_ALLOC_DS 158
3DSTATE_PUSH_CONSTANT_ALLOC_GS 160
3DSTATE_PUSH_CONSTANT_ALLOC_HS 162
3DSTATE_PUSH_CONSTANT_ALLOC_PS 164
3DSTATE_PUSH_CONSTANT_ALLOC_VS 166
3DSTATE_RASTER 168
3DSTATE_SAMPLE_MASK 174
3DSTATE_SAMPLE_PATTERN 176
3DSTATE_SAMPLE_PATTERN 182

3DSTATE_SAMPLER_PALETTE_LOAD0 188
3DSTATE_SAMPLER_PALETTE_LOAD1 189
3DSTATE_SAMPLER_STATE_POINTERS_DS 191
3DSTATE_SAMPLER_STATE_POINTERS_GS 192
3DSTATE_SAMPLER_STATE_POINTERS_HS 193
3DSTATE_SAMPLER_STATE_POINTERS_PS 194
3DSTATE_SAMPLER_STATE_POINTERS_VS 195
3DSTATE_SBE 196
3DSTATE_SBE_SWIZ 199
3DSTATE_SCISSOR_STATE_POINTERS 201
3DSTATE_SF 202
3DSTATE_SO_BUFFER 207
3DSTATE_SO_DECL_LIST 210
3DSTATE_STENCIL_BUFFER 213
3DSTATE_STREAMOUT 216
3DSTATE_TE 221
3DSTATE_URB_DS 225
3DSTATE_URB_GS 227
3DSTATE_URB_HS 229
3DSTATE_URB_VS 231
3DSTATE_VERTEX_BUFFERS 233
3DSTATE_VERTEX_ELEMENTS 235
3DSTATE_VF 237
3DSTATE_VF_INSTANCING 239
3DSTATE_VF_SGVS 241
3DSTATE_VF_STATISTICS 244
3DSTATE_VF_TOPOLOGY 245
3DSTATE_VIEWPORT_STATE_POINTERS_CC 246
3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP 247
3DSTATE_VS 248
3DSTATE_WM_CHROMAKEY 257
3DSTATE_WM_DEPTH_STENCIL 258
3DSTATE_WM 262

3DSTATE_WM_HZ_OP 268
A64 Byte Scattered Write MSD....274
A64 Dword Scattered Read MSD....275
A64 Dword Scattered Write MSD....276
A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD278
A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 280
A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD282
A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 284
A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD286
A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 288
A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD 290
A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD....292
A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD 294
A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD 296
A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD 298
A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD 300
A64 Hword Block Read MSD....302
A64 Hword Block Write MSD....303
A64 Oword Block Read MSD....304
A64 Oword Block Write MSD....305
A64 Oword Dual Block Read MSD 306
A64 Oword Dual Block Write MSD 307
A64 Oword Unaligned Block Read MSD 308
A64 Qword Scattered Write MSD 309
A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD311
A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 313
A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD315
A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 317
A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD319
A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 321
A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD 323
A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD 325
A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD 327

A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD 329
A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD 331
A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD 333
A64 Untyped Surface Read MSD 335
A64 Untyped Surface Write MSD 336
Addition 337
Addition with Carry 339
Arithmetic Shift Right 340
Average 342
Bit Field Extract 343
Bit Field Insert 1 346
Bit Field Insert 2 348
Bit Field Reverse 351
Branch Converging 352
Branch Diverging 354
Break 356
Byte Scattered Read MSD 358
Byte Scattered Write MSD 360
Call 362
Call Absolute 364
Compare 366
Compare NaN 368
Conditional Select 370
Conditional Send Message 372
Constant Cache Dword Scattered Read MSD 374
Constant Cache Oword Block Read MSD 376
Constant Cache Oword Dual Block Read MSD 378
Constant Cache Oword Unaligned Block Read MSD 380
Continue 381
Count Bits Set 383
Dot Product 2 384
Dot Product 3 385
Dot Product 4 387

Dot Product Homogeneous 388
Dword Atomic Counter Binary with Return Data Operation MSD 389
Dword Atomic Counter Binary Write Only Operation MSD....390
Dword Atomic Counter Unary with Return Data Operation MSD 391
Dword Atomic Counter Unary Write Only Operation MSD 392
Dword Scattered Read MSD....393
Dword Scattered Write MSD....395
Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD 397
Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD 398
Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD 399
Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD 400
Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD....401
Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD....402
Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD 403
Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD 404
Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD 405
Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD 406
Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD ...407
Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD 408
Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD..409
Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD 410
Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD....411
Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD 412
Dword Typed Atomic Integer Binary with Return Data Operation MSD....413
Dword Typed Atomic Integer Binary Write Only Operation MSD....414
Dword Typed Atomic Integer Trinary with Return Data Operation MSD 415
Dword Typed Atomic Integer Trinary Write Only Operation MSD....416
Dword Typed Atomic Integer Unary with Return Data Operation MSD....417
Dword Typed Atomic Integer Unary Write Only Operation MSD....418
Dword Untyped Atomic Integer Binary with Return Data Operation MSD 419
Dword Untyped Atomic Integer Binary Write Only Operation MSD....420
Dword Untyped Atomic Integer Trinary with Return Data Operation MSD 421
Dword Untyped Atomic Integer Trinary Write Only Operation MSD 422

Dword Untyped Atomic Integer Unary with Return Data Operation MSD 423
Dword Untyped Atomic Integer Unary Write Only Operation MSD 424
End If....425
Extended Math Function 427
Find First Bit from LSB Side....430
Find First Bit from MSB Side 431
Fraction....433
Goto 434
GPGPU_CSR_BASE_ADDRESS 436
GPGPU_WALKER 438
Half Precision HI8DS Render Target Write MSD 442
Half Precision LO8DS Render Target Write MSD 444
Half Precision REP16 Render Target Write MSD 446
Half Precision SIMD8 Render Target Write MSD 449
Half Precision SIMD16 Render Target Write MSD 451
Halt....454
HCP_BSD_OBJECT 456
HCP_IND_OBJ_BASE_ADDR_STATE 458
HCP_PIC_STATE 460
HCP_PIPE_BUF_ADDR_STATE 468
HCP_PIPE_MODE_SELECT 473
HCP_QM_STATE....476
HCP_REF_IDX_STATE....479
HCP_SLICE_STATE 482
HCP_SURFACE_STATE....489
HCP_TILE_STATE....491
HCP_WEIGHTOFFSET_STATE 495
HI8DS Render Target Write MSD 498
Illegal 500
Integer Subtraction with Borrow 501
Join....502
Jump Indexed....504
Leading Zero Detection 506

Line 507
Linear Interpolation 508
LO8DS Render Target Write MSD 510
Logic And 512
Logic Not 513
Logic Or 514
Logic Xor 515
MEDIA_CURBE_LOAD 516
MEDIA_INTERFACE_DESCRIPTOR_LOAD 518
MEDIA_OBJECT 520
MEDIA_OBJECT_GRPID 524
MEDIA_OBJECT_PRT 528
MEDIA_OBJECT_WALKER 530
MEDIA_STATE_FLUSH 537
MEDIA_VFE_STATE 539
Media Block Read MSD 547
Media Block Write MSD 548
Media Transpose Read MSD 549
Memory Fence MSD 550
MFC_AVC_PAK_OBJECT 551
MFC_JPEG_ HUFF_TABLE_STATE 553
MFC_JPEG_SCAN_OBJECT 555
MFC_MPEG2_PAK_OBJECT 558
MFC_MPEG2_SLICEGROUP_STATE 560
MFD_AVC_BSD_OBJECT 568
MFD_AVC_DPB_STATE 570
MFD_AVC_PICID_STATE 573
MFD_AVC_SLICEADDR 575
MFD_IT_OBJECT 577
MFD_JPEG_BSD_OBJECT 580
MFD_MPEG2_BSD_OBJECT 583
MFD_VC1_BSD_OBJECT 585
MFD_VC1_LONG_PIC_STATE 588

MFD_VC1_SHORT_PIC_STATE....602
MFD_VP8_BSD_OBJECT....611
MFX_AVC_DIRECTMODE_STATE....617
MFX_AVC_IMG_STATE....624
MFX_AVC_REF_IDX_STATE....642
MFX_AVC_SLICE_STATE....645
MFX_AVC_WEIGHTOFFSET_STATE....657
MFX_BSP_BUF_BASE_ADDR_STATE....660
MFX_DBK_OBJECT....668
MFX_FQM_STATE....677
MFX_IND_OBJ_BASE_ADDR_STATE....679
MFX_JPEG_HUFF_TABLE_STATE....694
MFX_JPEG_PIC_STATE....696
MFX_MPEG2_PIC_STATE....704
MFX_PAK_INSERT_OBJECT....718
MFX_PIPE_BUF_ADDR_STATE....722
MFX_PIPE_MODE_SELECT....742
MFX_QM_STATE....749
MFX_STATE_POINTER....751
MFX_STITCH_OBJECT....753
MFX_SURFACE_STATE....755
MFX_VC1_DIRECTMODE_STATE....762
MFX_VC1_PRED_PIPE_STATE....767
MFX_VP8_BSP_BUF_BASE_ADDR_STATE....773
MFX_VP8_Encoder_CFG....786
MFX_VP8_PAK_OBJECT....797
MFX_VP8_PIC_STATE....799
MFX_WAIT....831
MI_ARB_CHECK....832
MI_ARB_CHECK....833
MI_ARB_CHECK....834
MI_ARB_CHECK....835
MI_ARB_ON_OFF....836

MI_ATOMIC 838
MI_BATCH_BUFFER_END 844
MI_BATCH_BUFFER_END 845
MI_BATCH_BUFFER_END 846
MI_BATCH_BUFFER_END 847
MI_BATCH_BUFFER_START 848
MI_BATCH_BUFFER_START 851
MI_BATCH_BUFFER_START 853
MI_BATCH_BUFFER_START 855
MI_CLFLUSH 857
MI_CONDITIONAL_BATCH_BUFFER_END 859
MI_CONDITIONAL_BATCH_BUFFER_END 861
MI_CONDITIONAL_BATCH_BUFFER_END 863
MI_CONDITIONAL_BATCH_BUFFER_END 865
MI_COPY_MEM_MEM 867
MI_COPY_MEM_MEM 869
MI_COPY_MEM_MEM 871
MI_COPY_MEM_MEM 873
MI_DISPLAY_FLIP 876
MI_FLUSH_DW 881
MI_FLUSH_DW 885
MI_FLUSH_DW 888
MI_LOAD_REGISTER_IMM 891
MI_LOAD_REGISTER_IMM 893
MI_LOAD_REGISTER_IMM 895
MI_LOAD_REGISTER_IMM 897
MI_LOAD_REGISTER_MEM 899
MI_LOAD_REGISTER_REG 901
MI_LOAD_SCAN_LINES_EXCL 903
MI_LOAD_SCAN_LINES_EXCL 905
MI_LOAD_SCAN_LINES_INCL 907
MI_LOAD_SCAN_LINES_INCL 909
MI_LOAD_URB_MEM 911

MI_MATH 912
MI_MATH 913
MI_MATH 914
MI_MATH 915
MI_NOOP 916
MI_NOOP 917
MI_NOOP 918
MI_NOOP 919
MI_PREDICATE 920
MI_REPORT_HEAD 922
MI_REPORT_HEAD 923
MI_REPORT_HEAD 924
MI_REPORT_HEAD 925
MI_RS_CONTROL 926
MI_RS_STORE_DATA_IMM 928
MI_SEMAPHORE_SIGNAL 930
MI_SEMAPHORE_WAIT 933
MI_SEMAPHORE_WAIT 937
MI_SEMAPHORE_WAIT 940
MI_SEMAPHORE_WAIT 943
MI_SEMAPHORE_WAIT 946
MI_SET_CONTEXT 949
MI_SET_PREDICATE 952
MI_STORE_DATA_IMM 954
MI_STORE_DATA_IMM 956
MI_STORE_DATA_IMM 958
MI_STORE_DATA_IMM 961
MI_STORE_DATA_INDEX 964
MI_STORE_DATA_INDEX 966
MI_STORE_DATA_INDEX 968
MI_STORE_DATA_INDEX 970
MI_STORE_REGISTER_MEM 972
MI_STORE_URB_MEM 974

MI_SUSPEND_FLUSH 976
MI_SUSPEND_FLUSH 977
MI_SUSPEND_FLUSH 978
MI_SUSPEND_FLUSH 979
MI_TOPOLOGY_FILTER 980
MI_UPDATE_GTT 981
MI_UPDATE_GTT 982
MI_UPDATE_GTT 984
MI_UPDATE_GTT 985
MI_URB_ATOMIC_ALLOC 986
MI_URB_CLEAR 987
MI_USER_INTERRUPT 988
MI_USER_INTERRUPT 989
MI_USER_INTERRUPT 990
MI_USER_INTERRUPT 991
MI_WAIT_FOR_EVENT 992
MI_WAIT_FOR_EVENT 997
Move 1003
Move Indexed 1005
Multiply 1007
Multiply Accumulate 1009
Multiply Accumulate High 1011
Multiply Add 1013
Multiply Add for Macro 1016
No Operation 1019
Oword Block Read MSD 1020
Oword Block Write MSD 1021
Oword Dual Block Read MSD 1022
Oword Dual Block Write MSD 1023
Oword Unaligned Block Read MSD 1024
PIPE_CONTROL 1025
PIPELINE_SELECT 1033
Plane 1035

REP16 Render Target Write MSD 1037
Return 1039
Round Down 1040
Round to Nearest or Even 1041
Round to Zero 1042
Round Up 1043
Scattered Move 1044
Scratch Block Read MSD 1045
Scratch Block Write MSD 1047
Select 1048
Shift Left 1050
Shift Right 1051
SIMD8 Render Target Write MSD 1052
SIMD16 Render Target Write MSD 1054
STATE_BASE_ADDRESS 1056
STATE_PREFETCH 1064
STATE_SIP 1066
Sum of Absolute Difference 2 1067
Sum of Absolute Difference Accumulate 2 1068
SWTESS_BASE_ADDRESS 1069
Typed Surface Read MSD 1071
Typed Surface Write MSD 1072
Untyped Surface Read MSD 1073
Untyped Surface Write MSD 1074
URB Hword Dual Block Read MSD 1075
URB Hword Dual Block Write MSD 1076
URB Oword Block Write MSD 1077
URB Oword Dual Block Read MSD 1078
URB Oword Dual Block Write MSD 1079
VEBOX_STATE 1080
VEBOX_SURFACE_STATE 1085
Wait Notification 1091
While 1093

XY_COLOR_BLT 1095
XY_FULL_BLT 1097
XY_FULL_IMMEDIATE_PATTERN_BLT 1100
XY_FULL_MONO_PATTERN_BLT 1103
XY_FULL_MONO_PATTERN_MONO_SRC_BLT 1107
XY_FULL_MONO_SRC_BLT 1111
XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT 1115
XY_MONO_PAT_BLT 1118
XY_MONO_PAT_FIXED_BLT 1121
XY_MONO_SRC_COPY_BLT 1124
XY_MONO_SRC_COPY_IMMEDIATE_BLT 1127
XY_PAT_BLT 1130
XY_PAT_BLT_IMMEDIATE 1133
XY_PAT_CHROMA_BLT 1136
XY_PAT_CHROMA_BLT_IMMEDIATE 1139
XY_PIXEL_BLT 1142
XY_SCANLINES_BLT 1143
XY_SETUP_BLT 1144
XY_SETUP_CLIP_BLT 1147
XY_SETUP_MONO_PATTERN_SL_BLT 1148
XY_SRC_COPY_BLT 1151
XY_SRC_COPY_CHROMA_BLT 1154
XY_TEXT_BLT 1157
XY_TEXT_IMMEDIATE_BLT 1159

3DPRIMITIVE

3DPRIMITIVE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 3h 3DPRIMITIVE Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h 3DPRIMITIVE Format: OpCode
	15	**Reserved**
	14:13	**Reserved** Format: MBZ
	12	**Reserved**
	11	**Reserved**
	10	**Indirect Parameter Enable** Format: Enable If set, the values in DW 2-5 are ignored and replaced by the current values of the corresponding 3DPRIM_xxx MMIO registers: • 3DPRIM_VERTEX_COUNT (instead of DW2: VertexCountPerInstance)

3DPRIMITIVE

		• 3DPRIM_START_VERTEX (instead of DW3: StartVertexLocation) • 3DPRIM_INSTANCE_COUNT (instead of DW4: InstanceCount) • 3DPRIM_START_INSTANCE (instead of DW5: StartInstanceLocation) • 3DPRIM_BASE_VERTEX (instead of DW6: BaseVertexLocation) Indirect Parameter Enable and End Offset Enable shall not be ENABLED at the same time, or behavior is UNDEFINED.
	9	**UAV Coherency Required** Format: U1 SW will be required to set this bit if there is the possibility of sharing a UAV from a previous 3DPRIMITVE command. If set, this command may cause a flush due to UAV coherency requirements. If none of the shaders have UAV access enabled, then this bit is ignored.
	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.
	7:0	**DWord Length** Default Value: 5h Excludes DWord (0,1) Project: CHV, BSW Format: =n Total Length - 2
1	31:10	**Reserved** Format: MBZ
	9	**End Offset Enable** Format: Enable If set, the Vertex Count Per Instance field is IGNORED, and the VB0ENDOFFSET register is used to indirectly specify the vertex count by defining the amount of valid data in VB0. The following restrictions apply: • VB0 must be enabled for use • VertexAccessType = SEQUENTIAL • Start Vertex Location = 0 • Start Instance Location = 0 • Base Vertex Location = 0 Vertices are output until EndOffset is reached or exceeded in VB0. If EndOffset is reached or exceeded within the data associated with a vertex, that vertex is considered incomplete and will not be output. Partial objects will be discarded (as is normally done).

3DPRIMITIVE

		If clear, End Offset is ignored. Indirect Parameter Enable and End Offset Enable must not be ENABLED at the same time, or behavior is UNDEFINED.
	8	**Vertex Access Type** This field specifies how data held in vertex buffers marked as VERTEXDATA is accessed by Vertex Fetch. **Value** / **Name** / **Description** 0h / SEQUENTIAL / VERTEXDATA buffers are accessed sequentiallyRequiref if End Offset Enable is ENABLED. 1h / RANDOM / VERTEXDATA buffers are accessed randomly via an index obtained from the Index Buffer.
	7:6	**Reserved**
	5:0	**Primitive Topology Type** Format: 3D_Prim_Topo_Type [CHV, BSW] See table below for encoding, see 3D Overview for diagrams and general comments **Description** This field specifies the topology type of 3D primitive generated by this command. Note that a single primitive topology (list/strip/fan/etc.) can contain a number of basic objects (lines, triangles, etc.). This field is ignored. The topology type is specified via the 3DSTATE_VF_TOPOLOGY command.
2	31:0	**Vertex Count Per Instance** Format: U32 Count of vertices This field specifies how many vertices are to be generated for each instance of the primitive topology. If End Offset Enable is clear: Format = U32 count of vertices Range = [0, 2^32-1] (upper limit probably constrained by VB size) Ignored if End Offset Enable or Indirect Parameter Enable is ENABLED. **Programming Notes** • This per-instance value should specify a valid number of vertices for the primitive topology type. E.g., for 3DPRIM_TRILIST_ADJ, this field should specify a multiple of 6 vertices. However, in cases where too few or too many vertices are provided, the unused vertices will be silently discarded by the pipeline. • A 0 value is this field effectively makes the command a 'no-operation'.
3	31:0	**Start Vertex Location** Format: U32 structure index This field specifies the "starting vertex" for each instance. This allows skipping over part of the vertices in a buffer if, for example, a previous 3DPRIMITIVE command had already drawn the primitives associated with the earlier entries. For SEQUENTIAL access, this field specifies, for each instance, a starting structure index into the vertex buffers For RANDOM access, this field specifies, for each instance, a starting index into the Index Buffer.

3DPRIMITIVE

		Programming Notes • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED
4	31:0	**Instance Count** Format: U32 Count of instances This field specifies the number of instances by which the primitive topology is to be regenerated. A value of 0 indicates "no instances" (no-op operation). A value of 1 effectively specifies "non-instanced" operation, though vertex buffers will still be used to provide instance data, if so programmed. Ignored if Indirect Parameter Enable is ENABLED. **Value**: [0, FFFFFFFFh] — **Name**:
5	31:0	**Start Instance Location** Format: U32 structure index **Description** This field specifies the "starting instance" for the command as an initial structure index into Vertex Buffers for vertex elements with InstancingEnable set. Subsequent instances will access sequential instance data structures, as controlled by the Instance Data Step Rate. **Programming Notes** • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED.
6	31:0	**Base Vertex Location** Format: S31 index structure bias This field specifies a signed bias to be added to values read from the index buffer. This allows the same index buffer values to access different vertex data for different commands.This field applies only to RANDOM access mode. This field is ignored for SEQUENTIAL access mode, where there Start Vertex Location can be used to specify different regions in the vertex buffers. **Programming Notes** • Access of any data outside of the valid extent of a vertex or index buffer will return the value 0 (i.e., appears as if the data stored at the invalid location was 0). • Must be set to 0 if End Offset Enable is ENABLED. • Ignored if Indirect Parameter Enable is ENABLED.

3DSTATE_AA_LINE_PARAMETERS

3DSTATE_AA_LINE_PARAMETERS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ah 3DSTATE_AA_LINE_PARAMS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Default Value: 1h Excludes Dword (0,1) Project: All Format: =n Total Length - 2
1	31:24	**AA Point Coverage Bias** Project: CHV, BSW Format: U0.8 This field specifies the bias term to be used in the aa coverage computation for edges 0 and 3.
	23:16	**AA Coverage Bias** Project: All Format: U0.8

3DSTATE_AA_LINE_PARAMETERS

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

3DSTATE_BINDING_TABLE_EDIT_DS

3DSTATE_BINDING_TABLE_EDIT_DS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command edits the binding table for DS.

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

3DSTATE_BINDING_TABLE_EDIT_DS

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

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

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

Format:	BINDING_TABLE_EDIT_ENTRY [CHV, BSW]

3DSTATE_BINDING_TABLE_EDIT_GS

3DSTATE_BINDING_TABLE_EDIT_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command edits the binding table for GS.

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

3DSTATE_BINDING_TABLE_EDIT_GS

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

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

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

Format:	BINDING_TABLE_EDIT_ENTRY [CHV, BSW]

3DSTATE_BINDING_TABLE_EDIT_HS

3DSTATE_BINDING_TABLE_EDIT_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command edits the binding table for HS.

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

3DSTATE_BINDING_TABLE_EDIT_HS

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

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

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

Format:	BINDING_TABLE_EDIT_ENTRY [CHV, BSW]

3DSTATE_BINDING_TABLE_EDIT_PS

3DSTATE_BINDING_TABLE_EDIT_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command edits the binding table for PS.

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

3DSTATE_BINDING_TABLE_EDIT_PS

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

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

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

Format:	BINDING_TABLE_EDIT_ENTRY [CHV, BSW]

3DSTATE_BINDING_TABLE_EDIT_VS

3DSTATE_BINDING_TABLE_EDIT_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_BINDING_TABLE_EDIT_VS

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

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

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

Format:	BINDING_TABLE_EDIT_ENTRY [CHV, BSW]

3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_BINDING_TABLE_POINTERS_DS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 28h 3DSTATE_BINDING_TABLE_POINTERS_DS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:16	**Reserved** Project: All Format: MBZ
	15:5	**Pointer to DS Binding Table** Project: CHV, BSW Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When HW binding table is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When HW-generated binding table is enabled

3DSTATE_BINDING_TABLE_POINTERS_DS		
		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.
	4:0	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_BINDING_TABLE_POINTERS_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 29h 3DSTATE_BINDING_TABLE_POINTERS_GS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:16	**Reserved** Project: All Format: MBZ
	15:5	**Pointer to GS Binding Table** Project: CHV, BSW Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When HW binding table is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When HW-generated binding table is enabled

3DSTATE_BINDING_TABLE_POINTERS_GS		
		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.
	4:0	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_HS

3DSTATE_BINDING_TABLE_POINTERS_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 27h 3DSTATE_BINDING_TABLE_POINTERS_HS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:16	**Reserved** Project: All Format: MBZ
	15:5	**Pointer to HS Binding Table** Project: CHV, BSW Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When HW binding table is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When HW-generated binding table is enabled

3DSTATE_BINDING_TABLE_POINTERS_HS		
		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.
	4:0	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_PS

3DSTATE_BINDING_TABLE_POINTERS_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 2Ah 3DSTATE_BINDING_TABLE_POINTERS_PS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:16	**Reserved** Project: All Format: MBZ
	15:5	**Pointer to PS Binding Table** Project: CHV, BSW Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When HW binding table is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When HW-generated binding table is enabled

3DSTATE_BINDING_TABLE_POINTERS_PS

		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.
	4:0	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POINTERS_VS

3DSTATE_BINDING_TABLE_POINTERS_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 26h 3DSTATE_BINDING_TABLE_POINTERS_VS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:16	**Reserved** Project: All Format: MBZ
	15:5	**Pointer to VS Binding Table** Project: CHV, BSW Format: SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256 When HW binding table is disabled Format: SurfaceStateOffset[16:6]BINDING_TABLE_STATE*256 When HW-generated binding table is enabled

3DSTATE_BINDING_TABLE_POINTERS_VS

		Specifies an aligned address offset of the function's BINDING_TABLE_STATE. The offset's base and alignment differ depending on whether HW Binding Table is enabled: If HW Binding Table is disabled, the offset is relative to **Surface State Base Address** and the alignment is **32B**. If HW Binding Table is enabled the offset is relative to the **Binding Table Pool Base Address** and the alignment is **64B**.
	4:0	**Reserved** Project: All Format: MBZ

3DSTATE_BINDING_TABLE_POOL_ALLOC

3DSTATE_BINDING_TABLE_POOL_ALLOC

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_BINDING_TABLE_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Project: All Format: =n Value: 2h — Name: DWORD_COUNT_n **[Default]** — Project: CHV, BSW
1..2 **Project:** CHV,	63:12	**Binding Table Pool Base Address** Project: CHV, BSW

3DSTATE_BINDING_TABLE_POOL_ALLOC

BSW

Format:	GraphicsAddress[63:12]BindingTablePool

This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

11 — Binding Table Pool Enable

Project:	CHV, BSW
Format:	U1

When this bit is set it enables HW generation of binding tables. When this bit is cleared it disables HW generation of binding tables.

10 — Reserved

Project:	CHV, BSW

9:7 — Reserved

Project:	CHV, BSW

6:0 — Surface Object Control State

Project:	CHV, BSW
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for this surface.

Programming Notes
Bit 2 is not programmable and is always zero.

3 **Project:** CHV, BSW

31:12 — Binding Table Pool Buffer Size

Project:	CHV, BSW
Format:	U20

This field specifies the size of the buffer in 4K pages. Any access which straddle or go past the end of the buffer will return 0.

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

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

11 — Reserved

Project:	CHV, BSW
Format:	MBZ

10:0 — Reserved

Project:	CHV, BSW
Format:	MBZ

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

When the BLEND_STATE pointer changes but not the CC_STATE pointer, driver needs to force a CC_STATE pointer change to improve blend performance in pixel backend.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 24h 3DSTATE_BLEND_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:6	**Blend State Pointer** Project: All Format: DynamicStateOffset[31:6]BLEND_STATE*8 Specifies the 64-byte aligned offset of the BLEND_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved**

3DSTATE_BLEND_STATE_POINTERS

		Project: All Format: MBZ
	0	**Blend State Pointer Valid** Project: CHV, BSW Format: Enable This bit, if set, indicates that the BLEND_STATE pointer has changed and new state needs to be fetched.

3DSTATE_CC_STATE_POINTERS

3DSTATE_CC_STATE_POINTERS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

When the CC_STATE pointer changes but not the BLEND_STATE pointer, driver needs to force a BLEND_STATE pointer change in order to improve blend performance in the pixel backend.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Eh 3DSTATE_CC_STATE_POINTERS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n
1	31:6	**Color Calc State Pointer** Project: All Format: DynamicStateOffset[31:6]COLOR_CALC_STATE Specifies the 64-byte aligned offset of the COLOR_CALC_STATE. This offset is relative to the **Dynamic State Base Address**.
	5:1	**Reserved**

3DSTATE_CC_STATE_POINTERS

		Project: All
		Format: MBZ
	0	**Color Calc State Pointer Valid** Project: CHV, BSW Format: Enable If set, the hardware will fetch the CC state. This bit is context saved and restored so the CC state is considered undefined once this bit is cleared due to the possiblity of the CC state changing between context switches.

3DSTATE_CHROMA_KEY

3DSTATE_CHROMA_KEY

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 04h 3DSTATE_CHROMA_KEY Format: Opcode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:30	**ChromaKey Table Index** Project: All Format: U2 index Selects which entry in the ChromaKey table is to be loaded Value: [0,3] Name:

3DSTATE_CHROMA_KEY

	29:0	**Reserved** Project: All Format: MBZ
2	31:0	**ChromaKey Low Value** This field specifies the "low" (minimum) value of the chroma key range. Texel samples are considered "matching the key" if each component of the texel falls within the (inclusive) chroma range.See ChromaKey High Value for further format, programming info.
3	31:0	**ChromaKey High Value** This field specifies the "high" (maximum) value of the chroma key range. Texel samples are considered "matching the key" if each component of the texel falls within the (inclusive) chroma range.

Programming Notes

ChromaKey values are specified using 8-bit channels. When using surface formats with less than 8 bits per channel, the device will expand channels by replicating the required number of MSBs into the LSBs of each channel. Software must account for this conversion when it programs Chromakey Low/High Values (e.g., by performing the same replication).

For channels that do not exist in the actual surface (e.g., Alpha channel for non-ARGB maps), software must explicitly program full range high/low values (High=FFh, Low=0h for formats using unsigned chroma key values, High=7Fh, Low=FFh for formats using sign magnitude chroma key values) in order to effectively remove the comparison of that field from the ChromaKey function.

For channels in SNORM format in the surface format, the value in the high/low value for that channel is interpreted in sign magnitude format. Negative zero value is not supported (use positive zero instead). For channels with mixed UNORM/SNORM formats (i.e. R5G5_SNORM_B6_UNORM), the ChromaKey is programmed as if all channels are SNORM.

YUV ChromaKey will use an interpolated chrominance value from the map for comparison to the chroma key values for those texels without chrominance due to downsampling. The chrominance value used is the average of values to the left and right of the texel in question.

It is UNDEFINED to program any component of the ChromaKey High Value to be less than the corresponding component of ChromaKey Low Value.

Format = interpreted according to associated texel format "class":

Only the surface formats listed as supported for chroma key in the surface formats table can be used with this feature. Use of any other surface format with chroma key enabled is UNDEFINED.

Surface Format	31:24	23:15	16:8	7:0
ARGB and BC (DXT) formats	A	R	G	B
YCrCb formats	A	Cr	Y	Cb

3DSTATE_CLEAR_PARAMS

3DSTATE_CLEAR_PARAMS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description	Project
This command defines the depth clear value delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.	CHV, BSW

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

3DSTATE_CLEAR_PARAMS

		range of +0.0f and 1.0f inclusive; values outside this range are reserved.
2	31:1	**Reserved** Format: MBZ
	0	**Depth Clear Value Valid** Format: Boolean This field enables the **Depth Clear Value**. If clear, the depth clear value is obtained from interpolated depth of an arbitrary pixel of the primitive rendered with **Depth Buffer Clear** set in WM_STATE or 3DSTATE_WM. If set, the depth clear value is obtained from the **Depth Clear Value** field of this command.

3DSTATE_CLIP

3DSTATE_CLIP

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

3DSTATE_CLIP

	1h	Force	Forces Clip_INT::User Clip Distance Cull Test Enable Bitmask to use the value in 3DSTATE_CLIP:: User Clip Distance Cull Test Enable Bitmask

19 Vertex Sub Pixel Precision Select

Project:	All
Format:	U1

Selects the number of fractional bits maintained in the vertex data

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

18 Early Cull Enable

Project:	All
Format:	Enable

This field is used to enable/disable the EarlyCull function.

17 Force User Clip Distance Clip Test Enable Bitmask

Project:	All
Format:	Enable

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

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

16 Force Clip Mode

Format:	Enable

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

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

15:11 Reserved

Project:	All
Format:	MBZ

10 Clipper Statistics Enable

Project:	All
Format:	Enable

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

Value	Name	Description	Project

3DSTATE_CLIP

		0h	Disable	CL_INVOCATIONS_COUNT cannot increment	All
		1h	Enable	CL_INVOCATIONS_COUNT can increment	All
	9:8	**Reserved**			
		Project:			
		Format:			MBZ
	7:0	**User Clip Distance Cull Test Enable Bitmask**			
		Project:		All	
		Format:		Enable[8]	
		This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip).DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.			
2	31	**Clip Enable**			
		Project:		All	
		Format:		Enable	
		Specifies whether the Clip function is enabled or disabled (pass-through).			
	30	**API Mode**			
		Project:		All	
		Controls the definition of the NEAR clipping plane			
		Value	**Name**	**Description**	**Project**
		0h	OGL	NEAR VP boundary == 0.0 (NDC)	All
	29	**Reserved**			
		Project:		All	
		Format:		MBZ	
	28	**Viewport XY Clip Test Enable**			
		Project:		All	
		Format:		Enable	
		This field is used to control whether the Viewport X, Y extents are considered in VertexClipTest. See Tristrip Clipping subsection.			
	27	**Reserved**			
		Project:		All	
		Format:		MBZ	
	26	**Guardband Clip Test Enable**			
		Project:		All	
		Format:		Enable	
		This field is used to control whether the Guardband X, Y extents are considered in VertexClipTest			

3DSTATE_CLIP

for non-point objects. If the Guardband ClipTest is DISABLED but the Viewport XY ClipTest is ENABLED, ClipDetermination operates as if the Guardband were coincident with the Viewport. If both the Guardband and Viewport XY ClipTest are DISABLED, all vertices are considered "visible" with respect to the XY directions.

25:24 **Reserved**

Project:	All
Format:	MBZ

23:16 **User Clip Distance Clip Test Enable Bitmask**

Project:	All
Format:	Enable[8]

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

15:13 **Clip Mode**

Project:	All

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

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

12:10 **Reserved**

Project:	All
Format:	MBZ

9 **Perspective Divide Disable**

Project:	All
Format:	Disable

This field disables the Perspective Divide function performed on homogeneous position read from the URB. This feature can be used by software to submit pre-transformed "screen-space" geometry for rasterization. This likely requires the W component of positions to contain "rhw"

3DSTATE_CLIP

(aka 1/w) in order to support perspective-correct interpolation of vertex attributes. Likewise, the X, Y, Z components will likely be required to be X/W, Y/W, Z/W. Note that the device does not support clipping when perspective divide is disabled. Software must specify CLIPMODE_ACCEPT_ALL whenever it disables perspective divide. This implies that software must ensure that object positions are completely contained within the "guardband" screen-space limits imposed by the SF unit (e.g., by clipping in CPU SW before submitting the objects).

8 Non-Perspective Barycentric Enable

Project:	All
Format:	Enable

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

7:6 Reserved

Project:	All
Format:	MBZ

5:4 Triangle Strip/List Provoking Vertex Select

Project:	All
Format:	U2

enumerated type
This field selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex".

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

3:2 Line Strip/List Provoking Vertex Select

Project:	All
Format:	U2

enumerated type
This field selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".

Value	Name	Project
0h	0	All

3DSTATE_CLIP

		1h	1	All
		2h	Reserved	All
		3h	Reserved	All
	1:0	**Triangle Fan Provoking Vertex Select**		
		Project:		All
		Format:		U2
		enumerated type		
		This field selects which vertex of a triangle (in a triangle fan primitive) is considered the "provoking vertex".		
		Value	**Name**	**Project**
		0h	0	All
		1h	1	All
		2h	2	All
		3h	Reserved	All
3	31:28	**Reserved**		
		Project:		All
		Format:		MBZ
	27:17	**Minimum Point Width**		
		Project:	All	
		Format:	U8.3 pixels	
		This value is used to clamp read-back PointWidth values.		
	16:6	**Maximum Point Width**		
		Project:	All	
		Format:	U8.3 pixels	
		This value is used to clamp read-back PointWidth values.		
	5	**Force Zero RTA Index Enable**		
		Project:	All	
		Format:	Enable	
		If set, the Clip unit will ignore the read-back RTAIndex and operate as if the value 0 was read-back. If clear, the read-back value is used.		
	4	**Reserved**		
		Project:	All	
		Format:	MBZ	

3DSTATE_CLIP

	3:0	**Maximum VP Index** Project: All Format: U4-1 index value (# of viewports) This field specifies the maximum valid VPIndex value, corresponding to the number of active viewports. If the source of the VPIndex exceeds this maximum value, a VPIndex value of 0 is passed down the pipeline. Note that this clamping does not affect a VPIndex value stored in the URB.

3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT_DS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

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

Programming Notes	Project
[CHV, BSW] A 3DSTATE_GATHER_DS command must be dispatched along with any 3DSTATE_CONSTANT_DS command when Gather Pool is enabled within a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ah 3DSTATE_CONSTANT_DS Format: OpCode
	15	**Reserved** Project: CHV, BSW Format: MBZ
	14:8	**Constant Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command. Programming Notes: Constant Buffer Object Control State must be always programmed to zero. — Project: CHV, BSW
	7:0	**DWord Length** Project: All

3DSTATE_CONSTANT_DS

		Format: =n Total Length - 2 Value / Name / Project: 9h — Excludes DWord (0,1) **[Default]** — CHV, BSW
1..10 **Project:** CHV, BSW	319:0	**Constant Body** Project: CHV, BSW Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes	Project
[CHV, BSW]: A 3DSTATE_GATHER_GS command must be dispatched along with any 3DSTATE_CONSTANT_GS command when the Gather Pool is enabled within a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 16h 3DSTATE_CONSTANT_GS Format: OpCode
	15	**Reserved** Project: All Format: MBZ
	14:8	**Constant Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command. Programming Notes: Constant Buffer Object Control State must be always programmed to zero. — Project: CHV, BSW
	7:0	**DWord Length** Project: All

3DSTATE_CONSTANT_GS

		Format: =n Total Length - 2 **Value** / **Name** / **Project** 9h / Excludes DWord (0,1) **[Default]** / CHV, BSW
1..10 **Project:** CHV, BSW	319:0	**Constant Body** Project: CHV, BSW Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes	Project
A 3DSTATE_GATHER_HS command must be dispatched along with any 3DSTATE_CONSTANT_HS command when Gather Pool is enabled within a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 19h 3DSTATE_CONSTANT_HS Format: OpCode
	15	**Reserved** Project: CHV, BSW Format: MBZ
	14:8	**Constant Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command. Programming Notes: Constant Buffer Object Control State must be always programmed to zero. — Project: CHV, BSW
	7:0	**DWord Length** Project: All

3DSTATE_CONSTANT_HS

		Format: =n Total Length - 2 Value / Name / Project: 9h — Excludes DWord (0,1) **[Default]** — CHV, BSW
1..10 **Project:** CHV, BSW	319:0	**Constant Body** Project: CHV, BSW Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_PS

3DSTATE_CONSTANT_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes	Project
[CHV, BSW]: A 3DSTATE_GATHER_PS command must be dispatched along with any 3DSTATE_CONSTANT_PS command when the Gather Pool is enabled within a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 17h 3DSTATE_CONSTANT_PS Format: OpCode
	15	**Reserved** Project: CHV, BSW Format: MBZ
	14:8	**Constant Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command. Programming Notes: Constant Buffer Object Control State must be always programmed to zero. — Project: CHV, BSW
	7:0	**Dword Length** Project: All

3DSTATE_CONSTANT_PS

		Format: =n Total Length - 2 **Value** / **Name** / **Project** 9h / Excludes DWord (0,1) **[Default]** / CHV, BSW
1..10 **Project:** CHV, BSW	319:0	**Constant Body** Project: CHV, BSW Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_CONSTANT_VS

3DSTATE_CONSTANT_VS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

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

Programming Notes	Project
[CHV, BSW] A 3DSTATE_GATHER_VS command must be dispatched along with any 3DSTATE_CONSTANT_VS command when Gather Pool is enabled within a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 15h 3DSTATE_CONSTANT_VS Format: OpCode
	15	**Reserved** Project: CHV, BSW Format: MBZ
	14:8	**Constant Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for all constant buffers defined in this command. Programming Notes: Constant Buffer Object Control State must be always programmed to zero. — Project: CHV, BSW
	7:0	**DWord Length** Project: All

3DSTATE_CONSTANT_VS

		Format: =n Total Length - 2 **Value** / **Name** / **Project** 9h / Excludes DWord (0,1) **[Default]** / CHV, BSW
1..10 **Project:** CHV, BSW	319:0	**Constant Body** Project: CHV, BSW Format: 3DSTATE_CONSTANT(Body) Following table is the shared portion of the 3DSTATE_CONSTANT command for VS, HS, DS, and GS

3DSTATE_DEPTH_BUFFER

3DSTATE_DEPTH_BUFFER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 5h 3DSTATE_DEPTH_BUFFER Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 6h Excludes Dword (0,1) Format: =n Excludes DWord(0,1)
1	31:29	**Surface Type** Value / Name / Description: 0h / SURFTYPE_1D / Defines a 1-dimensional map or array of maps 1h / SURFTYPE_2D / Defines a 2-dimensional map or array of maps 2h / SURFTYPE_3D / Defines a 3-dimensional (volumetric) map 3h / SURFTYPE_CUBE / Defines a cube map 4h-6h / Reserved /

3DSTATE_DEPTH_BUFFER

7h	SURFTYPE_NULL	Defines a null surface

Programming Notes

The Surface Type of the depth buffer must be the same as the Surface Type of the render target(s) (defined in SURFACE_STATE), unless either the depth buffer or render targets are SURFTYPE_NULL.

28 Depth Write Enable

Format:	Enable

This field enables depth writes to the depth buffer surface. Both this field and the Depth Buffer Write Enable field in DEPTH_STENCIL_STATE must be enabled in order for depth writes to occur.

27 Stencil Write Enable

Format:	Enable

This field enables stencil writes to the depth buffer or stencil buffer surface, depending on where stencil is located. Both this field and the Stencil Buffer Write Enable field in DEPTH_STENCIL_STATE must be enabled in order for stencil writes to occur.

26:23 Reserved

Format:	MBZ

22 Hierarchical Depth Buffer Enable

Format:	Enable

If enabled, indicates that a hierarchical depth buffer is defined.

Programming Notes

If this field is enabled, the Software Tiled Rendering Mode must be NORMAL. This field must be disabled if Early Depth Test Enable is disabled OR if depth buffer surface type is NULL.

21 Reserved

Format:	MBZ

20:18 Surface Format

Specifies the format of the depth buffer. See Stencil Test Enable field in DEPTH_STENCIL_STATE field for restrictions on the use of some of these formats.

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

3DSTATE_DEPTH_BUFFER

17:0 Surface Pitch

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

Value	Name	Description
[7Fh,3FFFFh]		corresponding to [128B, 256KB] also restricted to a multiple of 128B

Programming Notes
The pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].

2..3 Project: CHV, BSW — 63:0 Surface Base Address

Project:	CHV, BSW
Format:	GraphicsAddress[63:0]DepthBuffer

This field specifies address of the buffer in mapped Graphics Memory. Graphics Address [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] = [47].

Programming Notes
The Depth Buffer can only be mapped to Main Memory (uncached).
If the buffer is linear, the surface must be 64-byte aligned.

4 — 31:18 Height

Format:	U14-1

This field specifies the height of the surface. If the surface is MIP-mapped, this field contains the height of the base MIP level.

Value	Name	Description	Exists If
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,16383]	Legal Range	Height of surface - 1 (y/v dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,2047]	Legal Range	Height of surface - 1 (y/v dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
[0,16383]	Legal Range	y/v dimension	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

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

17:4 Width

Format:	U14-1

3DSTATE_DEPTH_BUFFER

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

Value	Name	Description	Exists If
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D')
[0,2047]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
[0,16383]	Legal Range	Width of surface - 1 (x/u dimension)	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

Programming Notes

The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). For cube maps, Width must be set equal to Height. The Width of the depth buffer must be the same as the Width of the render target(s) (defined in SURFACE_STATE), unless Surface Type is SURFTYPE_1D or SURFTYPE_2D with Depth = 0 (non-array) and LOD = 0 (non-mip mapped).

3:0 **LOD**

Format:	U4 for LOD units

Value	Name
[0,14]	

Programming Notes

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

5 31:21 **Depth**

Format:	U11-1

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

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

3DSTATE_DEPTH_BUFFER

		Programming Notes The Depth of the depth buffer must be the same as the Depth of the render target(s) (defined in SURFACE_STATE).
	20:10	**Minimum Array Element** Format: U11 **For 1D and 2D Surfaces:** This field indicates the minimum array element that can be accessed as part of this surface. The delivered array index is added to this field before being used to address the surface. **For 3D Surfaces** This field indicates the minimum 'R' coordinate on the LOD currently being rendered to. This field is added to the delivered array index before it is used to address the surface. **For Other Surfaces** This field is ignored Value / Name / Exists If: [0,2047] SURFTYPE_1D/2D — (Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D'|Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D') [0,2047] SURFTYPE_3D — (Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_3D')
	9:7	**Reserved** Format: MBZ
	6:0	**Depth Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the depth buffer.
6	31:26	**Reserved** Project: CHV, BSW Format: MBZ
	25:0	**Reserved** Format: MBZ
7 **For 1D and 2D Surfaces:** This field must be set to the same value as the	31:21	**Render Target View Extent** Format: U11-1 Value / Name / Description / Exists If: [0,2047] Legal Range — Number of array elements- 1 — (Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_1D') [0,2047] Legal Range — Number of array elements- 1 — (Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_2D') [0,2047] Legal — To indication — (Structure[RENDER_SURFACE_STATE][Surface

3DSTATE_DEPTH_BUFFER

Depth field. **For 3D Surfaces:** This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to. **For Other Surfaces** This field is ignored.

	Range	extent of [1,2048]	Type]=='SURFTYPE_3D')
[0,0]	Legal Range	Must be zero	(Structure[RENDER_SURFACE_STATE][Surface Type]=='SURFTYPE_CUBE')

20:15 **Reserved**

14:0 **Surface QPitch**

Format:	QPitch[16:2]

Description
This field specifies the distance in rows between array slices. It is used only in the following cases: • **Surface Array** is enabled *OR* • **Number of Mulitsamples** is not NUMSAMPLES_1 and **Multisampled Surface Storage Format** set to MSFMT_MSS *OR* • **Surface Type** is SURFTYPE_CUBE
Other surface types: field is ignored

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

Programming Notes
Software must ensure that this field is set to a value sufficiently large that array slices in the surface do not overlap. Refer to the *Memory Data Formats* section for information on how surfaces are stored.

3DSTATE_DRAWING_RECTANGLE

3DSTATE_DRAWING_RECTANGLE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 00h 3DSTATE_DRAWING_RECTANGLE Format: OpCode
	15:14	**Core Mode Select** Project: CHV, BSW Format: U2 Specifies which core this command will be considered valid and update based on the state in this command. Value / Name / Description / Project 0h / Legacy / Both cores are enabled and will update the state. / All 1h / Core 0 Enabled / State will be updated in Core 0 only / All 2h / Core 1 Enabled / State will be updated in Core 1 only / All 3h / Reserved / / All
	13:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:16	**Clipped Drawing Rectangle Y Min**

3DSTATE_DRAWING_RECTANGLE

		Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) Specifies Ymin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with Y coordinates less than Ymin will be clipped out. **Value** / **Name** [0,16383] / Device ignores bits 31:30 **Programming Notes** This value can be larger than Clipped Drawing Rectangle Y Max. If Ymin>Ymax, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.
	15:0	**Clipped Drawing Rectangle X Min** Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) Specifies Xmin value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with X coordinates less than Xmin will be clipped out. **Value** / **Name** [0,16383] / Device ignores bits 15:14 **Programming Notes** This value can be larger than Clipped Drawing Rectangle X Max. If Xmin>Xmax, the clipped drawing rectangle is null, all polygons are discarded. If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.
2	31:16	**Clipped Drawing Rectangle Y Max** Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) Specifies Ymax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Ymax will be clipped out. **Value** / **Name** [0,16383] / Device ignores bits 31:30 **Programming Notes** This value can be less than Clipped Drawing Rectangle Y Min. If Ymax<Ymin, the clipped drawing rectangle is null, all polygons are discarded. If Ymin==Ymax, the clipped drawing rectangle is 1 pixel wide in the Y direction.
	15:0	**Clipped Drawing Rectangle X Max** Project: All Format: U16 in Pixels from Color Buffer origin (upper left corner) Specifies Xmax value of (inclusive) intersection of Drawing rectangle with the Color (Destination) Buffer, used for clipping. Pixels with coordinates greater than Xmax will be clipped out.

3DSTATE_DRAWING_RECTANGLE

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

Programming Notes	Project
This value can be less than Clipped Drawing Rectangle X Min. If Xmax<Xmin, the clipped drawing rectangle is null, all polygons are discarded.If Xmin==Xmax, the clipped drawing rectangle is 1 pixel wide in the X direction.	CHV, BSW

3 | 31:16 **Drawing Rectangle Origin Y**

Project:	All
Format:	S15 in Pixels from Color Buffer origin (upper left corner).

Description	Project
Range: [-16384,16383] (Bit 31 should be a sign extension)	CHV, BSW
Specifies Y origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.	

15:0 **Drawing Rectangle Origin X**

Project:	All
Format:	S15 in Pixels from Color Buffer origin (upper left corner).

Description	Project
Range: [-16384,16383] (Bit 15 should be a sign extension)	CHV, BSW
Specifies X origin of Drawing Rectangle (in whole pixels) relative to origin of the Color Buffer, used to map incoming (Draw Rectangle-relative) vertex positions to the Color Buffer space.	

3DSTATE_DS

3DSTATE_DS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The state used by DS is defined with this inline state packet

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

3DSTATE_DS

3 : 31 — Single Domain Point Dispatch

Project:	CHV, BSW
Format:	U1 Enumerated Type

This field can be used to force single domain point SIMD4x2 DS threads.
This field is ignored if **SIMD8 Dispatch Enable** is set.

Value	Name	Description
0h	Multiple	Dual domain point SIMD4x2 thread dispatches are allowed.
1h	Single	Single domain point SIMD4x2 thread dispatches are forced.

Workaround	Project
Workaround: The Single Domain Point Dispatch must always be set to 0.	CHV, BSW

3 : 30 — Vector Mask Enable

Project:	CHV, BSW
Format:	U1 Enumerated Type

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

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

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

3 : 29:27 — Sampler Count

Project:	CHV, BSW
Format:	U3

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

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

3DSTATE_DS

	4h	13-16 Samplers	between 13 and 16 samplers used

26 Reserved

Project:	CHV, BSW
Format:	MBZ

25:18 Binding Table Entry Count

Project:	CHV, BSW
Format:	U8

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

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

Value	Name
[0,255]	

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

17 Thread Dispatch Priority

Project:	CHV, BSW
Format:	U1 Enumerated Type

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

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

16 Floating Point Mode

Project:	CHV, BSW
Format:	U1 Enumerated Type

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

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

3DSTATE_DS

	15	**Reserved** Project: CHV, BSW Format: MBZ
	14	**Accesses UAV** Project: CHV, BSW Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment. **Programming Notes** This field must not be set when DS Function Enable is disabled.
	13	**Illegal Opcode Exception Enable** Project: CHV, BSW Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if DS Function Enable is DISABLED.
	12:8	**Reserved** Project: CHV, BSW Format: MBZ
	7	**Software Exception Enable** Project: CHV, BSW Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if DS Function Enable is DISABLED.
	6:0	**Reserved** Project: CHV, BSW Format: MBZ
4..5 **Project:** CHV, BSW	63:10	**Scratch Space Base Pointer** Project: CHV, BSW Format: GeneralStateOffset[63:10]ScratchSpace Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if DS Function Enable is DISABLED.

3DSTATE_DS

9:4 Reserved

Project:	CHV, BSW
Format:	MBZ

3:0 Per-Thread Scratch Space

Project:	CHV, BSW
Format:	U4 power of 2 Bytes over 1K Bytes

Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit.The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.
This field is ignored if DS Function Enable is DISABLED.

Value	Name
[0,11]	indicating [1K Bytes, 2M Bytes]

Programming Notes
This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.

6 **Project:** CHV, BSW

31:25 Reserved

Project:	CHV, BSW
Format:	MBZ

24:20 Dispatch GRF Start Register For URB Data

Project:	CHV, BSW
Format:	GRFRegister[4:0]

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

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

19:18 Reserved

Project:	CHV, BSW
Format:	MBZ

17:11 Patch URB Entry Read Length

Project:	CHV, BSW
Format:	U7

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

Value	Name
[0,64]	

3DSTATE_DS

10 — Reserved

Project:	CHV, BSW
Format:	MBZ

9:4 — Patch URB Entry Read Offset

Project:	CHV, BSW
Format:	U6

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

Value	Name
[0,63]	

3:0 — Reserved

Project:	CHV, BSW
Format:	MBZ

7 — Project: CHV, BSW

31 — Reserved

Project:	CHV, BSW
Format:	MBZ

30 — Reserved

Project:	CHV, BSW
Format:	MBZ

29:21 — Maximum Number of Threads

Project:	CHV, BSW
Format:	U9-1 Thread Count

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

Value	Name	Description	Project
[0,503]		indicating thread count of [1,504]	CHV, BSW
[0,79]		indicating thread count of [1,80]	CHV, BSW

20:11 — Reserved

Project:	CHV, BSW
Format:	MBZ

10 — Statistics Enable

Project:	CHV, BSW

3DSTATE_DS

	Format:	Enable

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

9:5 **Reserved**

Project:	CHV, BSW
Format:	MBZ

4 **Reserved**

Project:	CHV, BSW
Format:	MBZ

3 **SIMD8 Dispatch Enable**

Project:	CHV, BSW
Format:	Enable

This field is used to specify how DS threads are dispatched. The setting of this field must agree with how the DS kernel was compiled. If ENABLED, SIMD8 DS thread dispatches are performed. The **Single Domain Point Dispatch** field is ignored. If DISABLED, SIMD4x2 thread dispatches are performed. The **Single Domain Point Dispatch** field can be used to force single domain point dispatches.

2 **Compute W Coordinate Enable**

Project:	CHV, BSW
Format:	Enable

If ENABLED, the DS unit will (for each domain point) compute W = 1 - (U + V) and pass the result as a floating point value in the DS thread payload. If DISABLED, 0.0 will be passed. This field must only be ENABLED for the tessellation of TRI domains, where UVW coordinates are required. This field must be DISABLED for other domains (as they only require UV coordinates) otherwise the computed W coordinate is UNDEFINED. This field is ignored if DS Function Enable is DISABLED.

1 **Cache Disable**

Project:	CHV, BSW
Format:	Disable

This bit controls the operation of the DS Cache. This field is ignored if DS Function Enable is DISABLED. If the DS Cache is DISABLED and the DS Function is ENABLED, the DS Cache is not used and all incoming domain points will be passed to DS threads. If the DS Cache is ENABLED and the DS Function is ENABLED, incoming domain points that do not hit in the DS Cache will be passed to DS threads. The DS Cache is invalidated whenever the DS Cache becomes DISABLED , whenever the DS Function Enable toggles, and between patches.

3DSTATE_DS

	0	**Function Enable** Project: CHV, BSW Format: Enable If ENABLED, DS threads will be spawned to process incoming domain points which miss in the DS cache. If DISABLED, the DS stage goes into pass-through mode and performs no specific processing. This field is always used. **Programming Notes** The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.
8 **Project:** CHV, BSW	31:28	**Reserved** Project: CHV, BSW Format: MBZ
	27	**Reserved**
	26:21	**Vertex URB Entry Output Read Offset** Project: CHV, BSW Format: U6 Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB by SBE. **Value** [0,63] — **Name**
	20:16	**Vertex URB Entry Output Length** Project: CHV, BSW Format: U5 Specifies the amount of URB data written for each Vertex URB entry, in 256-bit register increments. **Value** [1,16] — **Name** **Programming Notes** This length does not include the vertex header.
	15:8	**User Clip Distance Clip Test Enable Bitmask** Project: CHV, BSW Format: Mask[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made. DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

3DSTATE_DS		
	7:0	**User Clip Distance Cull Test Enable Bitmask** Project: CHV, BSW Format: Mask[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command sets up the Gather Pool for Gather Buffers.

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode GFXPIPE_3D
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Bh 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 2h — Name: DWORD_COUNT_n **[Default]** — Project: CHV, BSW
1..2 **Project:** CHV, BSW	63:48	**Reserved** Project: CHV, BSW Format: MBZ
	47:13	**Dx9 Constant Buffer Pool Base Address** Project: CHV, BSW

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

		Format: GraphicsAddress[47:13]Dx9_Constant_Buffer_Pool Specifies the base address of the Dx9 Constant Buffer pool.
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10	**Dx9 Constant Buffer Pool Enable** Project: CHV, BSW Format: Enable When this bit is set it enables HW Dx9 constants buffers. When this bit is cleared it disables HW Dx9 constant buffers, the local bits for the constant buffers are cleared and the buffers will not be save or restored as part of context.
	9:7	**Reserved** Project: CHV, BSW Format: MBZ
	6:0	**Surface Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 2 is not programmable and is always zero.
3 **Project:** CHV, BSW	31:13	**Dx9 Constant Buffer Pool Buffer Size** Project: CHV, BSW Format: U19 This field specifies the size of the buffer in 8K pages. Any access which straddle or go past the end of the buffer will return 0. Note that BufferSize=0 indicates that there is no valid data in the buffer. **Restriction** Programming size of zero is illegal in the case that the pool is enabled.
	12:0	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_DX9_CONSTANTB_PS

3DSTATE_DX9_CONSTANTB_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command sets a DX9 constant Boolean register for PS.

Programming Notes

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

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

3DSTATE_DX9_CONSTANTB_PS

1	31:16	**Reserved** Project: All Format: MBZ
	15	**Global Constant Register** Project: All Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:4	**Reserved** Project: All Format: MBZ
	3:0	**Constant Register Index** Project: All Format: U4 This field specifies the index of 1st boolean to be updated.
2..n	31:0	**Entry** Format: DX9_CONSTANTB_ENTRY [CHV, BSW] The nth boolean to be updated.

3DSTATE_DX9_CONSTANTB_VS

3DSTATE_DX9_CONSTANTB_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command sets a DX9 constant Boolean register for PS.

Programming Notes

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

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

3DSTATE_DX9_CONSTANTB_VS		
	15	**Global Constant Register** Format: U1 When this bit is set the global constant register set will be updated. When this bit is clear the local constant register set will be updated.
	14:4	**Reserved** Format: MBZ
	3:0	**Constant Register Index** Format: U4 This field specifies the index of 1st boolean to be updated.
2..n	31:0	**Entry** Format: DX9_CONSTANTB_ENTRY [CHV, BSW] The nth boolean to be updated.

3DSTATE_DX9_CONSTANTF_PS

3DSTATE_DX9_CONSTANTF_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

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

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

3DSTATE_DX9_CONSTANTF_VS

3DSTATE_DX9_CONSTANTF_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

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

3DSTATE_DX9_CONSTANTF_VS

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

3DSTATE_DX9_CONSTANTI_PS

3DSTATE_DX9_CONSTANTI_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

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

3DSTATE_DX9_CONSTANTI_PS

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

3DSTATE_DX9_CONSTANTI_VS

3DSTATE_DX9_CONSTANTI_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

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

3DSTATE_DX9_CONSTANTI_VS

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

3DSTATE_DX9_GENERATE_ACTIVE_PS

3DSTATE_DX9_GENERATE_ACTIVE_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 42h 3DSTATE_DX9_GENERATE_ACTIVE_PS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Project: All Format: =n Total Length - 2
1	31:24	**Reserved** Project: All Format: MBZ

3DSTATE_DX9_GENERATE_ACTIVE_PS

23:13 **Pointer to PS Constant Buffer**

Project:	All
Format:	ConstantBufferOffset[23:13]BINDING_TABLE_STATE*

Specifies the 8KB aligned address offset of the PS function's Dx9 constant buffer. This offset is relative to the DX9 Constant buffer Base Address.

12 **DX9 Enable**

Project:	All
Format:	Enable
Format:	U1

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

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

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

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

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

11 **Clamp Enable**

Project:	All
Format:	Enable
Format:	U1

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

Programming Notes	Project
The clamping only affects the values written out to the constant buffer and not the on-die registers.	CHV, BSW

10:8 **Reserved**

Project:	CHV, BSW
Format:	MBZ

7:0 **Reserved**

Project:	All
Format:	MBZ

3DSTATE_DX9_GENERATE_ACTIVE_VS

3DSTATE_DX9_GENERATE_ACTIVE_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

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

3DSTATE_DX9_GENERATE_ACTIVE_VS

12 — DX9 Enable

Format:	Enable

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

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

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

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

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

11 — Clamp Enable

Format:	Enable

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

Programming Notes	Project
The clamping only affects the values written out to the constant buffer and not the on-die registers.	CHV, BSW

10:8 — Reserved

Project:	CHV, BSW
Format:	MBZ

7:0 — Reserved

Format:	MBZ

3DSTATE_DX9_LOCAL_VALID_PS

3DSTATE_DX9_LOCAL_VALID_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_DX9_LOCAL_VALID_PS		
10	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Local ConstantB Valid Bits** Project: CHV, BSW Format: U16 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.

3DSTATE_DX9_LOCAL_VALID_VS

3DSTATE_DX9_LOCAL_VALID_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_DX9_LOCAL_VALID_VS		
10 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Local ConstantB Valid Bits** Project: CHV, BSW Format: U16 Each bit field when set indicates that the corresponding local register is valid. When the bit is clear it indicates the local register is invalid.

3DSTATE_GATHER_CONSTANT_DS

3DSTATE_GATHER_CONSTANT_DS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

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

Programming Notes

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

The constant buffer block (group of aligned 16 binding table entries) must be set before this command is issued.

If the surface type is NULL, any fetch using the surface state base address is not bound by the size of the surface state and the fetch still occurs.

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

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

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

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

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

3DSTATE_GATHER_CONSTANT_DS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 37h 3DSTATE_GATHER_CONSTANT_DS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,128] / Range / 1-128 Entries
1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool

3DSTATE_GATHER_CONSTANT_DS

		Programming Notes SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Project: CHV, BSW Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Project: All Format: MBZ
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2:0	**Reserved** Project: All Format: MBZ
3..n	15:0	**Entry** Format: GATHER_CONSTANT_ENTRY [CHV, BSW]

3DSTATE_GATHER_CONSTANT_GS

3DSTATE_GATHER_CONSTANT_GS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes

- The HW generated binding table must be enabled to use this command.
- The constant buffer block (group of aligned 16 binding table entries) must be set before this command is issued.
- If the surface type is NULL, any fetch using the surface state base address is not bound by the size of the surface state and the fetch still occurs.
- The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.
- When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.
- If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.
- The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory: •Resource Streamer Flush = 1 •3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h. •3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

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

3DSTATE_GATHER_CONSTANT_GS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 35h 3DSTATE_GATHER_CONSTANT_GS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,128] / Range / 1-128 Entries
1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool.

3DSTATE_GATHER_CONSTANT_GS

		Programming Notes SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Project: CHV, BSW Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Project: All Format: MBZ
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2:0	**Reserved** Project: All Format: MBZ
3..n	15:0	**Entry** Format: GATHER_CONSTANT_ENTRY [CHV, BSW]

3DSTATE_GATHER_CONSTANT_HS

3DSTATE_GATHER_CONSTANT_HS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

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

Programming Notes

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

The constant buffer block (group of aligned 16 binding table entries) must be set before this command is issued.

If the surface type is NULL, any fetch using the surface state base address is not bound by the size of the surface state and the fetch still occurs.

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

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

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

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

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

3DSTATE_GATHER_CONSTANT_HS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 36h 3DSTATE_GATHER_CONSTANT_HS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,128] / Range / 1-128 Entries
1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool.

3DSTATE_GATHER_CONSTANT_HS

		Programming Notes SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.
	5	**Constant Buffer Dx9 Generate Stall** Project: CHV, BSW Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Reserved** Project: All Format: MBZ
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2:0	**Reserved** Project: All Format: MBZ
3..n	15:0	**Entry** Format: GATHER_CONSTANT_ENTRY [CHV, BSW]

3DSTATE_GATHER_CONSTANT_PS

3DSTATE_GATHER_CONSTANT_PS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes

- The HW generated binding table must be enabled to use this command.
- The constant buffer block (group of aligned 16 binding table entries) must be set before this command is issued.
- If the surface type is NULL, any fetch using the surface state base address is not bound by the size of the surface state and the fetch still occurs.
- The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.
- When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.
- If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.
- The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory: •Resource Streamer Flush = 1 •3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h. •3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

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

3DSTATE_GATHER_CONSTANT_PS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 38h 3DSTATE_GATHER_CONSTANT_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 1 / DWORD_COUNT_n **[Default]** / excludes DWords 0,1 [1,128] / Range / 1-128 Entries
1	31:16	**Constant Buffer Valid** Format: U16 This field specifies which of the 16 constant buffers are used in the push constant gather. If a bit is set it indicates the corresponding constant buffer is used. If a bit is clear it indicates the corresponding constant buffer is not used. If this field is zero it indicate that the gather buffer is not used.
	15:12	**Constant Buffer Binding Table Block** Format: U4 This field specifies the 16 entry block constant buffer in the binding table. The constant buffer entry block must be aligned on a 16 entry boundary.
	11:2	**Reserved** Format: MBZ
	1:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:6	**Gather Buffer Offset** Format: GatherBufferOffset[22:6] This field specifies the offset of the gather buffer within the Gather Pool. **Programming Notes** SW increments the offset by the size of the gather buffer in 512 bit units for each gather buffer generated.

3DSTATE_GATHER_CONSTANT_PS

	5	**Constant Buffer Dx9 Generate Stall** Project: CHV, BSW Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Constant Buffer Dx9 Enable** Format: Enable When this bit is set it indicates that the constant buffer is a HW generated Dx9 constant buffer. The resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization. Additionally the Dx9 constant buffers are a single buffer but larger than 4KB. Internally the HW will treat the DX9 buffer as 2 constant buffers. When this bit is enable only the 1st constant buffer valid bit is set. The 2nd constant buffer surface pointer will automatically be the 1st pointer + 4KB.
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2:0	**Reserved** Format: MBZ
3..n	15:0	**Entry** Format: GATHER_CONSTANT_ENTRY [CHV, BSW]

3DSTATE_GATHER_CONSTANT_VS

3DSTATE_GATHER_CONSTANT_VS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes
The HW generated binding table must be enabled to use this command.
The constant buffer block (group of aligned 16 binding table entries) must be set before this command is issued.
If the surface type is NULL, any fetch using the surface state base address is not bound by the size of the surface state and the fetch still occurs.
The length of the gather table is derived from the total length of the command. The command length is in DWords, but the gather table entries are 16 bits in length. If there is an unused odd entry at the end of the command the channel mask should be set to all 0s.
When a 3DSTATE_GATHER_CONSTANT_* command is used there must be a matching 3DSTATE_CONSTANT_*. Furthermore the 3DSTATE_CONSTANT_* must occur in the same order as the 3DSTATE_GATHER_CONSTANT_*. For example if a 3DSTATE_GATHER_CONSTANT_VS occurs before a 3DSTATE_GATHER_CONSTANT_PS, then the 3DSTATE_CONSTANT_VS must occur before the 3DSTATE_CONSTANT_PS.
If Gather pool is enabled, there must be a corresponding 3DSTATE_GATHER_CONSTANT command with any 3DSTATE_CONSTANT for any particular shader. To avoid any update to the Gather pool, and yet program the 3DSTATE_CONSTANT for a particular shader, send a 3DSTATE_GATHER_CONSTANT command with all valid bits set to zero.
The following commands must be executed after any 3DSTATE_GATHER_CONSTANT_* command that has Constant Buffer Valid greater than zero: •(N times, mininum number is 4) MI_RS_STORE_DATA_IMM –To force engine idle before executing the next instruction. Write must occur to address that will not corrupt memory: •Resource Streamer Flush = 1 •3DSTATE_GATHER_CONSTANT_* (Ensures correct timing of sync between resource streamer and render pipeline) •The Constant Buffer Valid field should be zero and the Dword length equal to 1h. •3DSTATE_CONSTANT_*: •All values match the previous 3DSTATE_CONSTANT_*

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

3DSTATE_GATHER_CONSTANT_VS

26:24 3D Command Opcode

Default Value:	0h 3DSTATE_PIPELINED
Format:	OpCode

23:16 3D Command Sub Opcode

Default Value:	34h 3DSTATE_GATHER_CONSTANT_VS
Format:	OpCode

15:8 Reserved

Format:	MBZ

7:0 DWord Length

Format:	=n

Total Length - 2

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

1 31:16 Constant Buffer Valid

Format:	U16

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

15:12 Constant Buffer Binding Table Block

Format:	U4

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

11:2 Reserved

Format:	MBZ

1:0 Reserved

Project:	CHV, BSW
Format:	MBZ

2 31:23 Reserved

Format:	MBZ

22:6 Gather Buffer Offset

Format:	GatherBufferOffset[22:6]

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

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

3DSTATE_GATHER_CONSTANT_VS

	5	**Constant Buffer Dx9 Generate Stall** Project: CHV, BSW Format: Enable When this bit is set the resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization.
	4	**Constant Buffer Dx9 Enable** Format: Enable When this bit is set it indicates that the constant buffer is a HW generated Dx9 constant buffer. The resource streamer will wait for the Dx9 constant buffer generator to be done before issuing this command to ensure buffer synchronization. Additionally the Dx9 constant buffers are a single buffer but larger than 4KB. Internally the HW will treat the DX9 buffer as 2 constant buffers. When this bit is enable only the 1st constant buffer valid bit is set. The 2nd constant buffer surface pointer will automatically be the 1st pointer + 4KB.
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2:0	**Reserved** Format: MBZ
3..n	15:0	**Entry** Format: GATHER_CONSTANT_ENTRY [CHV, BSW]

3DSTATE_GATHER_POOL_ALLOC

3DSTATE_GATHER_POOL_ALLOC

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command sets up the Gather Pool for Gather Buffers.

Programming Notes

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

The gather constants can only be enabled if the binding table generator is also enabled. This command must only be programmed when resource streamer is enabled thru batch buffer start and MI_RS_CONTROL has not disabled resource streamer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ah 3DSTATE_GATHER_POOL_ALLOC Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Value: 2h — Name: DWORD_COUNT_n **[Default]** — Project: CHV, BSW
1..2 **Project:** CHV, BSW	63:12	**Gather Pool Base Address** Project: CHV, BSW Format: GraphicsAddress[63:12]Gather_Pool

3DSTATE_GATHER_POOL_ALLOC

		GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Gather Pool Enable** Project: CHV, BSW Format: Enable When this bit is set it enables HW gathering of push constants. When this bit is cleared it disables HW gathering of push constants.
	10:7	**Reserved** Project: CHV, BSW Format: MBZ
	6:0	**Memory Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this surface. **Programming Notes** Bit 2 is not programmable and is always zero.
3 **Project:** CHV, BSW	31:12	**Gather Pool Buffer Size** Project: CHV, BSW Format: U20 This field specifies the size of the buffer in 4K pages. Any access which straddle or go past the end of the buffer will return undefined data. Note that BufferSize=0 indicates that there is no valid data in the buffer. **Value** / **Name**: [0,1048575] **Restriction** Programming size of zero is illegal in the case that the pool is enabled.
	11:0	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_GS

3DSTATE_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Controls the GS stage hardware.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 11h 3DSTATE_GS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 8h Excludes DWord (0,1) Format: =n
1..2	63:6	**Kernel Start Pointer** Project: All Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location (1st GEN4 core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.
	5:0	**Reserved** Project: All Format: MBZ
3	31	**Single Program Flow**

3DSTATE_GS

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

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

30 **Vector Mask Enable**

Format:	U1 Enumerated Type

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

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

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

29:27 **Sampler Count**

Format:	U3

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

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

26 **Reserved**

25:18 **Binding Table Entry Count**

Format:	U8

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

3DSTATE_GS

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

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

17 **Thread Dispatch Priority**

Specifies the priority of the thread for dispatch.

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

16 **Floating Point Mode**

Project:	All

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

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

15:14 **Reserved**

Format:	MBZ

13 **Illegal Opcode Exception Enable**

Format:	Enable

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

12 **Accesses UAV**

Format:	Enable

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

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

11 **Mask Stack Exception Enable**

Format:	Enable

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

10:8 **Reserved**

Format:	MBZ

7 **Software Exception Enable**

Format:	Enable

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

3DSTATE_GS

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

3DSTATE_GS

28:23 Output Vertex Size

Project:	All
Format:	U6

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

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

Programming Notes

Programming Restrictions: The vertex size must be programmed as a multiple of 32B units with the following exception: Rendering is disabled (as per SOL stage state) and the vertex size output by the GS thread is 16B. If rendering is enabled (as per SOL state) the vertex size must be programmed as a multiple of 32B units. In other words, the only time software can program a vertex size with an odd number of 16B units is when rendering is disabled.

22:17 Output Topology

Project:	All
Format:	3DPrimType

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

16:11 Vertex URB Entry Read Length

Project:	All

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

Programming Notes

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

10 Include Vertex Handles

Project:	All
Format:	Boolean

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

Programming Notes

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

9:4 Vertex URB Entry Read Offset

Project:	All
Format:	U6

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

3DSTATE_GS

3:0 Dispatch GRF Start Register For URB Data

Format:	U4

Specifies the starting GRF register number for the URB portion (Constant + Vertices) of the thread payload.

Value	Name	Description	Project
[0,15]		indicating GRF [R0, R15]	CHV, BSW

Programming Notes
If Include Vertex Handles is enabled (pull or hybrid handles case), then For simd4x2: For DUAL_OBJECT dispatch mode this field should be: (((2*numVerticesPerObject) + 8 - 1)/8) + 1 For SINGLE and DUAL_INSTANCE dispatch modes this field should be: ((numVerticesPerObject +8 - 1)/8) + 1 If Include Primitive ID is set, then add 1 to the value obtained by using the above
For simd8: For InstanceCount == 1: numVerticesPerObject + 2 For InstanceCount > 1: 3

7 31:24 Maximum Number of Threads

Format:	U8/2-1 Thread Count

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

Value	Name	Description
[3,251]		indicating thread count of [8,504]
[3,39]		indicating thread count of [8,80]

Programming Notes
Note that this "Maximum Number of Threads" field is different from the other FF stages in that only an even number of threads.

23:20 Control Data Header Size

Format:	U4

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

Value	Name
[0,8]	32B Units

3DSTATE_GS

19:15 Instance Control

Format:	U5-1 #Instances

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

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

14:13 Default Stream Id

Format:	U2

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

12:11 Dispatch Mode

Format:	U2

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

Value	Name	Description	Programming Notes
0h	Single	Each thread shades a single instance of one object.	
1h	Dual Instance	Each thread shades possibly two instances of one object. If the InstanceCount is odd, a trailing dispatch of only one instance will be made for each object received. Not recommended if InstanceCount = 1, assuming a kernel optimized for SINGLE or DUAL_OBJECT dispatch would outperform a kernel compiled for DUAL_INSTANCE but only passed one instance.	
2h	Dual Object	Each thread shades one instance of possibly two objects. The GS unit attempt to pair objects together into one dispatch, but under some circumstances only one object may be dispatched (as controlled by the DispatchMask generated by the GS unit). Not valid for objects with more than 16 vertices per object. Not valid if InstanceCount > 1 (more than one instance per object).	
3h	SIMD8	Each thread shades up to 8 different objects or (if InstanceCount >1) 8 instances of a single object.	Not valid for objects with more than 6 vertices per object.

3DSTATE_GS

Programming Notes
The GS must be allocated at least two URB handles or behavior is UNDEFINED for Dual Instance or Dual Object mode.
At least 8 URB entries must be allocated in order to use SIMD8 DispatchMode.

10 **Statistics Enable**

Format:	Enable

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

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

9:5 **Invocations Increment Value**

Format:	U5

Specifies how much to increment the GS_INVOCATIONS_COUNT for each instance of each object. This control is provided to allow software to process multiple instances (from an API POV) in a single kernel invocation. In SINGLE dispatch mode, the counter will increment by this value for each dispatch (as it's only one instance of one object). In DUAL_INSTANCE mode, the counter will be incremented by the value if only one instance is included in the dispatch (i.e., the last odd instance), otherwise the counter will be incremented by twice this value. In DUAL_OBJECT dispatch mode, the counter will be incremented by the value if only one object is included in the dispatch (i.e., a forced dispatch of one object), otherwise the counter will be incremented by twice this value.

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

4 **Include Primitive ID**

Format:	Boolean

If set, R1 of the payload is written with Primitive ID value(s). If clear, these Primitive ID values are not included in the payload R1.

3 **Hint**

Format:	U1

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

3DSTATE_GS

2 **Reorder Mode**

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

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

Programming Notes
Workaround: reorder mode must be set to REORDER_LEADING and reordering must be done in the Geometry shader.
Workaround
Workaround: To work around a CHV, BSW issue, reorder mode must be set to REORDER_LEADING when GS is disabled.

1 **Discard Adjacency**

Format:	Enable

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

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

3DSTATE_GS

0 — Enable

Format:	Enable

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

8 — 31 — Control Data Format

Format:	U1

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

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

8 — 30 — Static Output

Format:	Enable

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

8 — 29:27 — Reserved

Format:	MBZ

8 — 26:16 — Static Output Vertex Count

Format:	U11 Count of object vertices

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

8 — 15:9 — Reserved

Format:	MBZ

8 — 8:0 — Reserved

Format:	MBZ

9 — 31:28 — Reserved

Format:	MBZ

9 — 27 — Reserved

9 — 26:21 — Vertex URB Entry Output Read Offset

Format:	U6

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

Value	Name
[0,63]	

3DSTATE_GS

	20:16	**Vertex URB Entry Output Length** Format: U5 Specifies the amount of URB data written for each Vertex URB entry, in 256-bit register increments. **Value** / **Name** [1,16] **Programming Notes** This length does not include the vertex header.
	15:8	**User Clip Distance Clip Test Enable Bitmask** Format: Enable[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept / must clip determination needs to be made. DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.
	7:0	**User Clip Distance Cull Test Enable Bitmask** Format: Enable[8] This 8 bit mask field selects which of the 8 user clip distances against which trivial reject / trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances.

3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_HIER_DEPTH_BUFFER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description	Project
This command sets the surface state of the hierarchical depth buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.	CHV, BSW

Programming Notes

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

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

3DSTATE_HIER_DEPTH_BUFFER

1	31:25	**Hierarchical Depth Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the hierarchical depth buffer.
	24:23	**Reserved** Project: CHV, BSW Format: MBZ
	22:17	**Reserved** Format: MBZ
	16:0	**Surface Pitch** Format: U17-1 Pitch in Bytes This field specifies the pitch of the hierarchical depth buffer in (#Bytes - 1). **Value** / **Name** [127, 1FFFFh] corresponding to [128B, 128KB] also restricted to a multiple of 128B **Programming Notes** Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].
2..3	63:0	**Surface Base Address** Format: GraphicsAddress[63:0]HierarchicalDepthBuffer This field specifies the address of the buffer in Graphics Memory. **Programming Notes** The Hierarchical Depth Buffer can only be mapped to Main Memory (uncached).
4	31:15	**Reserved** Format: MBZ
	14:0	**Surface QPitch** Project: CHV, BSW Format: QPitch[16:2] **Description** This field specifies the distance in rows between array slices. It is used only in the following cases: • **Surface Array** is enabled *OR* • **Number of Mulitsamples** is not NUMSAMPLES_1 and **Multisampled Surface Storage Format** set to MSFMT_MSS *OR* • **Surface Type** is SURFTYPE_CUBE

3DSTATE_HIER_DEPTH_BUFFER

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

Programming Notes

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

3DSTATE_HS

3DSTATE_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Controls the HS stage hardware.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Bh 3DSTATE_HS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Format: =n Value 7 — Name: Excludes DWord (0,1) **[Default]** — Project: CHV, BSW
1	31:30	**Reserved** Project: All Format: MBZ
	29:27	**Sampler Count** Project: All Format: U3 Specifies how many samplers (in multiples of 4) the HS kernels use. Used only for prefetching the associated sampler state entries.

3DSTATE_HS

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

26 **Reserved**

Project:	All
Format:	MBZ

25:18 **Binding Table Entry Count**

Project:	All
Format:	U8

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

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

17 **Thread Dispatch Priority**

Project:	CHV, BSW

Specifies the priority of the thread for dispatch

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

16 **Floating Point Mode**

Project:	All

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

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

15:14 **Reserved**

Project:	All
Format:	MBZ

3DSTATE_HS

	13	**Illegal Opcode Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.
	12	**Software Exception Enable** Project: CHV, BSW Format: Enable This bit gets loaded into EU CRO1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.
	11:8	**Reserved** Project: All Format: MBZ
	7:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31	**Enable** Project: All Format: Enable Specifies whether the HS function is enabled or disabled (pass-through). If ENABLED MI_TOPOLOGY_FILTER must be used to silently discard any topologies that the HS kernel is not expecting. E.g., if the HS kernel is expecting PATCHLIST_32 topologies, MI_TOPOLOGY_FILTER must be set to PATCHLIST_32 so only those topologies can reach the enabled HS. **Programming Notes** The tessellation stages (HS, TE and DS) must be enabled/disabled as a group. I.e., draw commands can only be issued if all three stages are enabled or all three stages are disabled, otherwise the behavior is UNDEFINED.
	30	**Reserved** Format: MBZ
	29	**Statistics Enable** Project: All Format: Enable This bit controls whether HS-unit-specific statistics register(s) will increment (for each patch).
	28:18	**Reserved** Project: All Format: MBZ

3DSTATE_HS

	17	**Reserved** Project: CHV, BSW Format: MBZ
	16:8	**Maximum Number of Threads** Project: CHV, BSW Format: U9-1 Specifies the maximum number of simultaneous threads allowed to be active. Used to avoid using up the scratch space. Programming the value of the max threads over the number of threads based off number of threads supported in the execution units may improve performance since the architecture allows threads to be buffered between the check for max threads and the actual dispatch into the EU. Programming the max values to a number less than the number of threads supported in the execution units may reduce performance.
	7:4	**Reserved** Format: MBZ
	3:0	**Instance Count** Format: U4-1 This field determines the number of threads (minus one) spawned per input patch. If the HS kernel uses a barrier function, software must restrict the **Instance Count** to the number of threads that can be simultaneously active within a subslice. Factors which must be considered includes scratch memory availability.
3..4 **Project:** CHV, BSW	63:6	**Kernel Start Pointer** Project: CHV, BSW Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location (1st GEN core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the Instruction Base Address.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ

Value	Name	Description	Project
[0,503]		indicating thread count of [1,504]	CHV, BSW
[0,79]		indicating thread count of [1,80]	CHV, BSW

Value	Name	Description
[0,15]		representing [1,16] instances

Programming Notes
A pipe_control with cs stall must be sent whenever the HS_STATE.InstanceCount changes from 0 (no instancing) to >0 (instancing) or when there is transition from HS_STATE.Enabled = false to (HS_STATE.Enabled = true && InstanceCount > 0).

3DSTATE_HS

5..6 **Project:** CHV, BSW

63:10 **Scratch Space Base Pointer**

Project:	CHV, BSW
Format:	GeneralStateOffset[63:10]

Value	Name	Description
[0,31]		Specifies the location of the scratch space area allocated to this FF unit, specified as a 1KB-granular offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space.

9:4 **Reserved**

Project:	CHV, BSW
Format:	MBZ

3:0 **Per-Thread Scratch Space**

Project:	CHV, BSW
Format:	U4 power of 2 Bytes over 1K Bytes

Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space.

Value	Name	Description
[0,11]		Indicating[1K Bytes, 2M Bytes

7 **Project:** CHV, BSW

31:29 **Reserved**

Project:	CHV, BSW
Format:	MBZ

28 **Reserved**

Format:	MBZ

27 **Single Program Flow**

Project:	CHV, BSW
Format:	Enable

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

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

26 **Vector Mask Enable**

Project:	CHV, BSW
Format:	U1 Enumerated Type

3DSTATE_HS

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

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

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

25 **Accesses UAV**

Project:	CHV, BSW
Format:	Enable

This field must be set when HS has a UAV access

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

24 **Include Vertex Handles**

Project:	CHV, BSW
Format:	Boolean

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

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

23:19 **Dispatch GRF Start Register For URB Data**

Project:	CHV, BSW
Format:	U5

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

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

18:17 **Reserved**

Project:	CHV, BSW
Format:	MBZ

3DSTATE_HS

	16:11	**Vertex URB Entry Read Length** Project: CHV, BSW Format: U6 Specifies the amount of URB data read and passed in the thread payload for each Vertex URB entry, in 256-bit register increments. This field is ignored if HS Function Enable is DISABLED. **Value** [0,63] **Name** **Programming Notes** **Programming Restriction:** This field must be a non-zero value if **Include Vertex Handles** is cleared to zero.
	10	**Reserved** Project: CHV, BSW Format: MBZ
	9:4	**Vertex URB Entry Read Offset** Project: CHV, BSW Format: U6 Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB before being included in the thread payload. This offset applies to all Vertex URB entries passed to the thread. This field is ignored if HS Function Enable is DISABLED. **Value** [0,63] **Name**
	3:1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Reserved** Project: CHV, BSW Format: MBZ
8 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_INDEX_BUFFER

3DSTATE_INDEX_BUFFER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_INDEX_BUFFER

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

3DSTATE_LINE_STIPPLE

3DSTATE_LINE_STIPPLE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_LINE_STIPPLE

	29:21	**Current Repeat Counter** Project: All Format: U9 This field sets the HW-internal repeat counter state. SW must initilize it to 1 if the modify enable is set.
	20	**Reserved** Project: All Format: MBZ
	19:16	**Current Stipple Index** Project: All Format: U4 This field sets the HW-internal stipple pattern index. SW must initialize it to 0 if the modify enable is set.
	15:0	**Line Stipple Pattern** Project: All Format: 16 bit mask Bit 15 = most significant bit, Bit 0 = least significant bit Specifies a pattern used to mask out bit specific pixels while rendering lines.
2	31:15	**Line Stipple Inverse Repeat Count** Project: All Format: U1.16 Range: [0.00390625, 1.0] Specifies the inverse (truncated) of the repeat count for the line stipple function.
	14:9	**Reserved** Project: All Format: MBZ
	8:0	**Line Stipple Repeat Count** Project: All Format: U9 Specifies the repeat count for the line stipple function. Value: [1, 256] Name:

3DSTATE_MONOFILTER_SIZE

3DSTATE_MONOFILTER_SIZE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_MONOFILTER_SIZE

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

3DSTATE_MULTISAMPLE

3DSTATE_MULTISAMPLE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Dh 3DSTATE_MULTISAMPLE Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Default Value: 0h Project: All Format: =n Total Length - 2 Excludes Dword (0,1)
1	31:6	**Reserved** Project: All Format: MBZ

3DSTATE_MULTISAMPLE

5 **Pixel Position Offset Enable**

Project:	CHV, BSW
Format:	U1

Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions.

Programming Notes
Setting this field along with setting the Pixel Location to upper left and number of multisamples to greater than one will cause the device to offset pixel postions by 0.5 both in horizontal and vertical directions. It is to be noted this is done to adjust the pixel co-ordinate system to DX9 like, so any WM_HZ_OP screen space rectangles (eg: legacy HiZ Clear, Resolve etc) generated internally by driver in this mode needs to be aware of this offset adjustment and send the rectangles according to alignment restriction taking this offset adjustment into consideration. SW can choose to set this bit only for DX9 API. DX10/OGL API's should not have any effect by setting or not setting this bit.

4 **Pixel Location**

Project:	All
Format:	U1

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

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

Programming Notes	Project
The programming of this field is assumed to be a function of the API being supported. Specifically, it is expected that OpenGL and DX10+ APIs require CENTER selection, while DX9- APIs require UL_CORNER selection.	
When 3DSTATE_RASTER::**ForcedSampleCount** is other than NUMRASTSAMPLES_0, this field must be 0h.	CHV, BSW

3:1 **Number of Multisamples**

Project:	All
Format:	U3

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

Value	Name	Description	Project
0h	1	1 sample/pixel	All
1h	2	2 samples/pixel	CHV, BSW
2h	4	4 samples/pixel	All
3h	8	8 samples/pixel	CHV, BSW
5h-7h	Reserved		All

3DSTATE_MULTISAMPLE

		Programming Notes The setting of this field must match the **Number of Multisamples** field in SURFACE_STATE of all bound render targets.
	0	**Reserved** Project: All Format: MBZ

3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_OFFSET

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_POLY_STIPPLE_OFFSET

	7:5	**Reserved** Project: All Format: MBZ
	4:0	**Polygon Stipple Y Offset** Project: All Format: U5 Specifies a 5 bit y address offset in the poly stipple pattern Value: [0,31] — Name:

3DSTATE_POLY_STIPPLE_PATTERN

3DSTATE_POLY_STIPPLE_PATTERN

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_PS_BLEND

3DSTATE_PS_BLEND

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

3DSTATE_PS_BLEND

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

3DSTATE_PS

3DSTATE_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 20h 3DSTATE_PS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0Ah Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1..2	63:6	**Kernel Start Pointer 0** Project: All Format: InstructionBaseOffset[63:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel[0]. This pointer is relative to the **Instruction Base Address.**
	5:0	**Reserved** Project: All Format: MBZ
3	31	**Single Program Flow** Project: All

3DSTATE_PS

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

Value	Name	Description	Project
0h	Multiple	Multiple Program Flows	All
1h	Single	Single Program Flows	All

30 **Vector Mask Enable**

Project:	All
Format:	U1 Enumerated Type

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

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

29:27 **Sampler Count**

Project:	All

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

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

26 **Single Precision Denormal Mode**

Project:	All

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

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

25:18 **Binding Table Entry Count**

Project:	All

Description

3DSTATE_PS

Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. **Note:** For kernels using a large number of binding table entries, it may be advantageous to set this field to zero to avoid prefetching too many entries and thrashing the state cache. This field is ignored if [PS Function Enable] is DISABLED.

When [HW Generated Binding Table] bit is enabled: This field indicates which cache lines (512bit units - 32 Binding Table Entry section) should be fetched. Each bit in this field corresponds to a cache line. Only the 1st 4 non-zero Binding Table entries of each 32 Binding Table entry section prefetched will have its surface state prefetched. See 3D Pipeline for more information.

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

17 **Thread Dispatch Priority**

Project:	All

Specifies the priority of the thread for dispatch.

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

16 **Floating Point Mode**

Project:	All

Specifies the floating point mode used by the dispatched thread.

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

15:14 **Rounding Mode**

Project:	All

Specifies the rounding mode used by the dispatched thread.

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

13 **Illegal Opcode Exception Enable**

Project:	All
Format:	Enable

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

3DSTATE_PS

	12	**Reserved** Project: All Format: MBZ
	11	**Mask Stack Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[12] (note the bit # difference). See Exceptions and ISA Execution Environment.
	10:8	**Reserved** Project: All Format: MBZ
	7	**Software Exception Enable** Project: All Format: Enable This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment.
	6:0	**Reserved** Project: All Format: MBZ
4..5	63:10	**Scratch Space Base Pointer** Project: All Format: GeneralStateOffset[63:10]ScratchSpace Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the General State Base Address. **Programming Notes** Scratch Space per slice is computed based on 4 sub-slices. SW must allocate scratch space enough so that each slice has 4 slices allowed.
	9:4	**Reserved** Project: All Format: MBZ
	3:0	**Per Thread Scratch Space** Project: All Format: U4 Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the Maximum Number of Threads each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space.

3DSTATE_PS

		Value / **Name**: [0,11] / indicating [1k bytes, 2M bytes] in powers of two
6	31:23	**Maximum Number of Threads Per PSD** Project: CHV, BSW Format: U8-2 Representing Thread Count **Description**: Range = [0, 62] --> [2, 64] threads. Specifies the maximum number of simultaneous threads allowed to be active per PSD. Depending on the GT mode register, Number of PSDs in the system determine maximum number PS threads per configuration. Has 2 PSDs therefore actual range for max PS threads is [4,128]. **Programming Notes**: If this field is changed between 3DPRIMITIVE commands, a PIPE_CONTROL command with **Stall at Pixel Scoreboard** set is required to be issued.
	22:12	**Reserved** Project: All Format: MBZ
	11	**Push Constant Enable** Project: All Format: Enable This field must be enabled if the sum of the **PS Constant Buffer [3:0] Read Length** fields in 3DSTATE_CONSTANT_PS is nonzero, and must be disabled if the sum is zero.
	10	**Reserved** Project: All Format: MBZ
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8	**Render Target Fast Clear Enable** Project: All Format: Enable This field is set to enable fast clear of the bound render targets. See "Render Target Fast Clear" for restrictions on enabling this field.
	7	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_PS

6 **Render Target Resolve Enable**

Project:	CHV, BSW
Format:	Enable

This field is set to enable clear value resolve on non-multisampled render targets. See "Render Target Resolve" for restrictions on enabling this field.

5 **Reserved**

Project:	
Format:	MBZ

4:3 **Position XY Offset Select**

Project:	All
Format:	U2 Enumerated Type

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

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

Programming Notes

SW Recommendation: If the PS kernel needs the Position Offsets to compute a Position XY value, this field should match Position ZW Interpolation Mode to ensure a consistent position.xyzw computation

If the PS kernel does not need the Position XY Offsets to compute a Position Value, then this field should be programmed to POSOFFSET_NONE, as the PS kernel should be using the various barycentric inputs to evaluate other-than-position attributes. However, this field can be used to pass Centroid or Sample offsets in the payload for special test modes (e.g., where barycentric coordinates are computed in the PS vs. being HW-generated and passed in the payload).

MSDISPMODE_PERSAMPLE is required in order to select POSOFFSET_SAMPLE.

2 **32 Pixel Dispatch Enable**

Project:	All
Format:	Enable

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

3DSTATE_PS

	1	**16 Pixel Dispatch Enable** Project: All Format: Enable Enables the Windower to dispatch 4 subspans in one payload. Variable Pixel Dispatch in Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations.
	0	**8 Pixel Dispatch Enable** Project: All Format: Enable Enables the Windower to dispatch 2 subspans from 1 object (polygon) in one payload. Variable Pixel Dispatch in Section: Pixel Grouping (Dispatch size) control for valid pixel dispatch combinations. **Programming Notes** / **Project** When Render Target Fast Clear Enable is ENABLED or Render Target Resolve Type = RESOLVE_PARTIAL or RESOLVE_FULL, this bit must be DISABLED. / CHV, BSW
7	31:23	**Reserved** Project: All Format: MBZ
	22:16	**Dispatch GRF Start Register For Constant/Setup Data 0** Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[0]. **Value** / **Name** [0,127] /
	15	**Reserved** Project: All Format: MBZ
	14:8	**Dispatch GRF Start Register For Constant/Setup Data 1** Project: All Format: U7 Specifies the starting GRF register number for the Constant/Setup portion of the thread payload for kernel[1]. **Value** / **Name** [0,127] /
	7	**Reserved** Project: All Format: MBZ

3DSTATE_PS

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

3DSTATE_PS_EXTRA

3DSTATE_PS_EXTRA

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4fh 3DSTATE_PS_EXTRA Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31	**Pixel Shader Valid** Project: All Format: Enable When set indicates a valid pixel shaderWhen this bit clear the rest of this command should also be clear.
	30	**Pixel Shader Does not write to RT** Project: All Format: Enable When set indicates the pixel shader does not write to render target.

3DSTATE_PS_EXTRA

29 oMask Present to Render Target

Project:	All
Format:	Enable

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

28 Pixel Shader Kills Pixel

Project:	All
Format:	Enable

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

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

27:26 Pixel Shader Computed Depth Mode

Project:	All
Format:	U2 Enumerated Type

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

Value	Name	Description
0h	PSCDEPTH_OFF	Pixel shader does not compute depth
1h	PSCDEPTH_ON	Pixel shader computes depth with no guarantee as to its value
2h	PSCDEPTH_ON_GE	Pixel shader computes depth and guarantees that oDepth >= SourceDepth
3h	PSCDEPTH_ON_LE	Pixel shader computes depth and guarantees that oDepth <= SourceDepth

Programming Notes

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

25 Reserved

24 Pixel Shader Uses Source Depth

Project:	All
Format:	Enable

This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be passed in the payload. The source depth value is interpolated according to the Position ZW Interpolation Mode state.

3DSTATE_PS_EXTRA

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

3DSTATE_PS_EXTRA

		Enabled.
	5	**Reserved** Project: CHV, BSW Format: MBZ
	4	**Reserved** Project: CHV, BSW Format: MBZ
	3	**Reserved** Project: CHV, BSW Format: MBZ
	2	**Pixel Shader Has UAV** Project: All Format: Enable Format: U1 Enumerated Type This field when set indicates that the pixel shader has a UAV attached to it.
	1	**Pixel Shader Uses Input Coverage Mask** Project: CHV, BSW Format: Enable This bit, if ENABLED, indicates that the PS kernel requires the input coverage mask to be passed in the payload.
	0	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes

Programming Restriction:

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

DWord	Bit	Description
0 **Programming Notes:** Workaround: This command must be followed by a PIPE_CONTROL with CS Stall bit set.	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_PUSH_CONSTANT_ALLOC_DS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

3DSTATE_PUSH_CONSTANT_ALLOC_DS

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

3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_PUSH_CONSTANT_ALLOC_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

- The sum of the Constant Buffer Offset and the Constant Buffer Size may not exceed the maximum value of the Constant Buffer Size.
- The sum of the constant length programmed in 3DSTATE_CONSTANT_GS must be equal or smaller then the size of the allocated space in the URB including the buffering for half cachelines.
- The 3DSTATE_CONSTANT_GS must be reprogrammed prior to the next 3DPRIMITIVE command after programming the 3DSTATE_PUSH_CONSTANT_ALLOC_GS.

See Push Constant URB Allocation section for more details.

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

3DSTATE_PUSH_CONSTANT_ALLOC_GS

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

3DSTATE_PUSH_CONSTANT_ALLOC_HS

3DSTATE_PUSH_CONSTANT_ALLOC_HS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes

Programming Restriction:

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 13h 3DSTATE_PUSH_CONSTANT_ALLOC_HS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

3DSTATE_PUSH_CONSTANT_ALLOC_HS

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

3DSTATE_PUSH_CONSTANT_ALLOC_PS

3DSTATE_PUSH_CONSTANT_ALLOC_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

Restriction:

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

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

3DSTATE_PUSH_CONSTANT_ALLOC_PS

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

3DSTATE_PUSH_CONSTANT_ALLOC_VS

3DSTATE_PUSH_CONSTANT_ALLOC_VS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

Programming Notes

Programming Restriction:

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

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

3DSTATE_PUSH_CONSTANT_ALLOC_VS

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

3DSTATE_RASTER

3DSTATE_RASTER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 50h 3DSTATE_RASTER Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 03h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:28	**Reserved** Project: All Format: MBZ
	27	**Reserved** Project: CHV, BSW Format: MBZ
	26:24	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_RASTER

23:22 **API Mode**

Project:	All

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

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

21 **Front Winding**

Project:	All

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

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

20:18 **Forced Sample Count**

Project:	All
Format:	U3 Enumerated Type

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

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

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

17:16 **Cull Mode**

Project:	All
Format:	3D_CullMode

3DSTATE_RASTER

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

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

Programming Notes

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

15 **Reserved**

Project:	All
Format:	MBZ

14 **Force Multisampling**

Project:	All

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

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

13 **Smooth Point Enable**

Project:	CHV, BSW
Format:	Enable

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

12 **DX Multisample Rasterization Enable**

Project:	All
Format:	Enable

Software sets this according to the API's multisample enable

Programming Notes

3DSTATE_RASTER

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

11:10 **DX Multisample Rasterization Mode**

Project:	All
Format:	U2 enumerated type

This field determines whether multisample rasterization is turned on/off, and how the pixel sample point(s) are defined. Software sets this according to the API's multisample state setting (if any)

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

Programming Notes

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

9 **Global Depth Offset Enable Solid**

Project:	All
Format:	Enable

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

8 **Global Depth Offset Enable Wireframe**

Project:	All
Format:	Enable

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

7 **Global Depth Offset Enable Point**

Project:	All
Format:	Enable

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

6:5 **Front Face Fill Mode**

Project:	All

3DSTATE_RASTER

Format:	U2 enumerated type

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

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

4:3 **Back Face Fill Mode**

Project:	All
Format:	U2 enumerated type

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

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

2 **Antialiasing Enable**

Project:	All
Format:	Enable

This field enables "alpha-based" line antialiasing.

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

1 **Scissor Rectangle Enable**

Project:	All
Format:	Enable

Enables operation of Scissor Rectangle.

3DSTATE_RASTER		
	0	**Viewport Z Clip Test Enable** Project: CHV, BSW Format: Enable This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.
2	31:0	**Global Depth Offset Constant** Format: IEEE_Float Specifies the constant term in the Global Depth Offset function.
3	31:0	**Global Depth Offset Scale** Format: IEEE_Float Specifies the scale term used in the Global Depth Offset function.
4	31:0	**Global Depth Offset Clamp** Format: IEEE_Float Specifies the clamp term used in the Global Depth Offset function.

3DSTATE_SAMPLE_MASK

3DSTATE_SAMPLE_MASK

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 18h 3DSTATE_DX9_LOCAL_VALID_PS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Format: =n Total Length - 2
1	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Sample Mask** Project: CHV, BSW Format: 16 bit mask Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_MULTISAMPLE) A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection. This mask must be ignored for centroid selection when RTIR is enabled i.e. Forced_Sample_Count > 0.

3DSTATE_SAMPLE_MASK

		Programming Notes • If **Number of Multisamples** is NUMSAMPLES_1, bits 15:1 of this field will be zeroed by HW. • If N**Number of Multisamples** is NUMSAMPLES_2, bits 15:2 of this field will be zeroed by HW. • If **Number of Multisamples** is NUMSAMPLES_4, bits 15:4 of this field will be zeroed by HW. • If **Number of Multisamples** is NUMSAMPLES_8, bits 15:8 of this field will be zeroed by HW.

3DSTATE_SAMPLE_PATTERN

3DSTATE_SAMPLE_PATTERN

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

The 3DSTATE_SAMPLE_PATTERN command is used to specify the sample offsets for all multisample sample modes. The set of offset used will be selected based on the multisample mode. This is non-pipelined state.

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_SAMPLE_PATTERN Format: OpCode
	15:8	**Reserved**
	7:0	**Dword Length** Default Value: 7 Format: =n Total Length - 2 Excludes Dword (0,1)
1..4	31:0	**Reserved**
5	31:28	**8x Sample7 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

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

3DSTATE_SAMPLE_PATTERN

	3:0	**8x Sample4 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
6	31:28	**8x Sample3 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample3 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample2 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	19:16	**8x Sample2 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	15:12	**8x Sample1 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	11:8	**8x Sample1 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample0 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample0 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
7	31:28	**4x Sample3 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	27:24	**4x Sample3 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	23:20	**4x Sample2 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

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

3DSTATE_SAMPLE_PATTERN

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

3DSTATE_SAMPLE_PATTERN

3DSTATE_SAMPLE_PATTERN

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The 3DSTATE_SAMPLE_PATTERN command is used to specify the sample offsets for all multisample sample modes. The set of offset used will be selected based on the multisample mode. This is non-pipelined state.

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_SAMPLE_PATTERN Format: OpCode
	15:8	**Reserved**
	7:0	**Dword Length** Default Value: 3 Project: All Format: =n Total Length - 2 Excludes Dword (0,1)
1	31:28	**8x Sample7 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	27:24	**8x Sample7 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 7 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample6 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	19:16	**8x Sample6 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 6 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	15:12	**8x Sample5 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample5 Y Offset** Format: U0.4 Subpixel Y offset of Sample 5 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	7:4	**8x Sample4 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample4 Y Offset** Project: All Format: U0.4

3DSTATE_SAMPLE_PATTERN

		Subpixel Y offset of Sample 4 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
2	31:28	**8x Sample3 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	27:24	**8x Sample3 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	23:20	**8x Sample2 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	19:16	**8x Sample2 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	15:12	**8x Sample1 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	11:8	**8x Sample1 Y Offset** Format: U0.4 Subpixel Y offset of Sample 1 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

	7:4	**8x Sample0 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
	3:0	**8x Sample0 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 0 relative to the UL pixel origin for 8x mode. Range: [0,0.9375]
3	31:28	**4x Sample3 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	27:24	**4x Sample3 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 3 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	23:20	**4x Sample2 X Offset** Project: All Format: U0.4 Subpixel X offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]
	19:16	**4x Sample2 Y Offset** Project: All Format: U0.4 Subpixel Y offset of Sample 2 relative to the UL pixel origin for 4x mode. Range: [0,0.9375]

3DSTATE_SAMPLE_PATTERN

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

3DSTATE_SAMPLE_PATTERN

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

3DSTATE_SAMPLER_PALETTE_LOAD0

3DSTATE_SAMPLER_PALETTE_LOAD0

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description

The 3DSTATE_SAMPLER_PALETTE_LOAD0 instruction is used to load 32-bit values into the first texture palette. The texture palette is used whenever a texture with a paletted format (containing "Px [palette0]") is referenced by the sampler.

This instruction is used to load all or a subset of the 256 entries of the first palette. Partial loads always start from the first (index 0) entry.

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

3DSTATE_SAMPLER_PALETTE_LOAD1

3DSTATE_SAMPLER_PALETTE_LOAD1

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

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

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

3DSTATE_SAMPLER_PALETTE_LOAD1

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

3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SAMPLER_STATE_POINTERS_HS

3DSTATE_SAMPLER_STATE_POINTERS_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SAMPLER_STATE_POINTERS_PS

3DSTATE_SAMPLER_STATE_POINTERS_PS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SAMPLER_STATE_POINTERS_VS

3DSTATE_SAMPLER_STATE_POINTERS_VS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SBE

3DSTATE_SBE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Fh 3DSTATE_SBE Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 02h Excludes DWord (0,1) Project: CHV, BSW Format: =n Total Length - 2
1	31:30	**Reserved** Format: MBZ
	29	**Force Vertex URB Entry Read Length** Project: CHV, BSW Format: Enable This field provides a work around override for the computation of SBE_INT::Vertex URB Entry Read Length. If asserted, 3DSTATE_SBE::Vertex URB Entry Read Length is be used directly. Otherwise, SBE_INT::Vertex URB Entry Read Length is computed normally.
	28	**Force Vertex URB Entry Read Offset** Project: CHV, BSW

3DSTATE_SBE

Format:	Enable	

This field provides a work around override for the computation of SBE_INT::Vertex URB Entry Read Offset. If asserted, 3DSTATE_SBE::Vertex URB Entry Read Offset is be used directly. Otherwise, SBE_INT::Vertex URB Entry Read Offset is computed normally.

27:22 **Number of SF Output Attributes**

Project:	CHV, BSW
Format:	U6 count of attributes

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

Value	Name
[0,32]	

21 **Attribute Swizzle Enable**

Format:	Enable

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

20 **Point Sprite Texture Coordinate Origin**

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

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

19 **Primitive ID Override Component W**

Project:	CHV, BSW
Format:	Enable

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

18 **Primitive ID Override Component Z**

Project:	CHV, BSW
Format:	Enable

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

17 **Primitive ID Override Component Y**

Project:	CHV, BSW
Format:	Enable

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

3DSTATE_SBE

	16	**Primitive ID Override Component X** Project: CHV, BSW Format: Enable If set, the X component of output attribute selected by Primitive ID Override Attribute Select is overridden with the Primitive ID.

	15:11	**Vertex URB Entry Read Length**
		Format: U5
		Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments.
		Value: [1,16] — Name:
		Programming Notes
		It is UNDEFINED to set this field to 0 indicating no Vertex URB data to be read.This field should be set to the minimum length required to read the maximum source attribute. The maximum source attribute is indicated by the maximum value of the enabled Attribute # Source Attribute if Attribute Swizzle Enable is set, Number of Output Attributes-1 if enable is not set.
		read_length = ceiling((max_source_attr+1)/2)

	10:5	**Vertex URB Entry Read Offset** Project: CHV, BSW Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB.
	4:0	**Primitive ID Override Attribute Select** Project: CHV, BSW Specifies which attribute is overridden w/ the Primitive ID **Programming Notes** Set all Primitive ID Override Component Select X/Y/Z/W to 0 to indicate there is no Primitive ID override.
2	31:0	**Point Sprite Texture Coordinate Enable** Format: Enable[32] When processing point primitives, the attributes from the incoming point vertex are typically copied to the point object corner vertices. However, if a bit is set in this field, the corresponding Attribute is selected as a Point Sprite Texture Coordinate, in which case each corner vertex is assigned a pre-defined texture coordinate as defined by the Point Sprite Texture Coordinate Origin state bit. Bit 0 corresponds to output Attribute 0.
3	31:0	**Constant Interpolation Enable** Format: Enable[32] This field is a bitmask containing a Constant Interpolation Enable bit for each corresponding attribute. If a bit is set, that attribute will undergo constant interpolation, and the corresponding WrapShortest Enable bits (if defined) will be ignored. If a bit is clear, components which are not enabled for WrapShortest interpolation (if defined) will be linearly interpolated.

3DSTATE_SBE_SWIZ

3DSTATE_SBE_SWIZ		
Project:	CHV, BSW	
Source:	RenderCS	
Length Bias:	2	
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 51h 3DSTATE_SBE_SWIZ Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 9h Excludes DWord (0,1) Format: =n Total Length - 2
1..8	15:0	**Attribute** Format: SF_OUTPUT_ATTRIBUTE_DETAIL
9..10	63:60	**Attribute 15 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	59:56	**Attribute 14 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	55:52	**Attribute 13 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	51:48	**Attribute 12 Wrap Shortest Enables** Format: WRAP_SHORTEST_ENABLE
	47:44	**Attribute 11 Wrap Shortest Enables**

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

3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_SCISSOR_STATE_POINTERS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

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

3DSTATE_SF

3DSTATE_SF

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

3DSTATE_SF

		Programming Notes Software must not program a value of 0.0 when running in MSRASTMODE_ON_xxx modes - zero-width lines are not available when multisampling rasterization is enabled. **Restriction** / **Project** Line widths higher than 40.0 will appear to be thinner than the programmed line width at certain angles. Software must expose the line widths higher than 40 to APIs only if the loss in quality is acceptable. — CHV, BSW Software must set the "WM_CHIKEN1" register bits "1 and 3" when rendering the AA lines higher than Line width of "40.0". — CHV, BSW
	11	**Legacy Global Depth Bias Enable** Format: Enable Enables the SF to use the Global Depth Offset Constant state unmodified. If this bit is not set, the SF will scale the Global Depth Offset Constant as described in section Error! Reference source not found. of this document. **Programming Notes** This bit should be set whenever non zero depth bias (Slope, Bias) values are used. Setting this bit may have some degradation of performance for some workloads.
	10	**Statistics Enable** Project: All Format: Enable If ENABLED, this FF unit will increment CL_PRIMITIVES_COUNT on behalf of the CLIP stage. If DISABLED, CL_PRIMITIVES_COUNT will be left unchanged. **Programming Notes** This bit should be set whenever clipping is enabled and the Statistics Enable bit is set in CLIP_STATE. It should be cleared if clipping is disabled or Statistics Enable in CLIP_STATE is clear.
	9:2	**Reserved** Format: MBZ
	1	**Viewport Transform Enable** Format: Enable This bit controls the Viewport Transform function.
	0	**Reserved** Format: MBZ
2	31:29	**Reserved** Format: MBZ
	28	**Reserved**
	27:18	**Reserved**

3DSTATE_SF

	17:16	**Line End Cap Antialiasing Region Width** Format: U2 This field specifies the distances over which the coverage of anti-aliased line end caps are computed. **Value** / **Name** / **Description** 0h / 0.5 pixels / 0.5 pixels 1h / 1.0 pixels / 1.0 pixels 2h / 2.0 pixels / 2.0 pixels 3h / 4.0 pixels / 4.0 pixels
	15	**Reserved** Format: MBZ
	14	**Reserved** Format: MBZ
	13	**Reserved**
	12	**Reserved**
	11:0	**Reserved** Format: MBZ
3	31	**Last Pixel Enable** Format: Enable If ENABLED, the last pixel of a diamond line will be lit. This state will only affect the rasterization of Diamond lines (will not affect wide lines or anti-aliased lines). **Programming Notes** Last pixel is applied to all lines of a LINELIST, and only the last line of a LINESTRIP.
	30:29	**Triangle Strip/List Provoking Vertex Select** Format: 0-based vertex index Selects which vertex of a triangle (in a triangle strip or list primitive) is considered the "provoking vertex". Used for flat shading of primitives.Does current implementation send provoking vertex first? **Value** / **Name** 0h / 0 1h / 1 2h / 2 3h / Reserved
	28:27	**Line Strip/List Provoking Vertex Select** Project: All Format: 0-based vertex index Selects which vertex of a line (in a line strip or list primitive) is considered the "provoking vertex".

3DSTATE_SF

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

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

Format:	0-based vertex index

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

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

24:15 **Reserved**

Format:	MBZ

14 **AA Line Distance Mode**

Format:	U1

This bit controls the distance computation for antialiased lines.

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

13 **Smooth Point Enable**

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

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

12 **Vertex Sub Pixel Precision Select**

Format:	U1

Selects the number of fractional bits maintained in the vertex data

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

3DSTATE_SF

	11	**Point Width Source** Controls whether the point width passed on the vertex or from state is used for rendering point primitives.

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

	10:0	**Point Width**

Format:	U8.3

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

3DSTATE_SO_BUFFER

3DSTATE_SO_BUFFER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Programming Notes	Project
Foreach SO Buffer, the 3DSTATE_SO_BUFFER must only be sent once prior to each 3DPRIMITIVE command.	CHV, BSW

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

3DSTATE_SO_BUFFER

	30:29	**SO Buffer Index** Format: U2 Specifies which of the four SO Buffers is being defined.
	28:22	**SO Buffer Object Control State** Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the SO buffer.
	21	**Stream Offset Write Enable** Format: Enable When set, this field allows the hardware to write SO_WRITE_OFFSET[Buffer#] as specified in the Stream Offset field. **Programming Notes** The field is operates irrespective of whether SO Buffer Enable is set or clear.
	20	**Stream Output Buffer Offset Address Enable** Format: Enable When set, this field allows the hardware to read/write the stream output buffer offset as specified in the "Stream Output Buffer Offset Address" field. **Programming Notes** The field is operates irrespective of whether SO Buffer Enable is set or clear.
	19:12	**Reserved** Format: MBZ
	11:0	**Reserved** Format: MBZ
2..3	63:48	**Reserved** Format: MBZ
	47:2	**Surface Base Address** Format: GraphicsAddress[47:2]SurfaceBase This field specifies the starting DWord address of the buffer in Graphics Memory.
	1:0	**Reserved** Format: MBZ
4	31:30	**Reserved** Format: MBZ
	29:0	**Surface Size** Format: U30-1 This field specifies the size of buffer in number DWords minus 1 of the buffer in Graphics Memory.

<table>
<tr><th colspan="3">3DSTATE_SO_BUFFER</th></tr>
<tr><td rowspan="3">5..6</td><td>63:48</td><td>Reserved
Format: MBZ</td></tr>
<tr><td>47:2</td><td>Stream Output Buffer Offset Address
Format: GraphicsAddress[47:2]OutputBuffer
This field specifies the high 16 bits of address of the buffer in Graphics Memory where the Stream Output Buffer Offset is stored when all the data has been written. It is also used to fetch the stream Output buffer Offset when needed.</td></tr>
<tr><td>1:0</td><td>Reserved
Format: MBZ</td></tr>
<tr><td>7</td><td>31:0</td><td>Stream Offset
This field specifies the Offset in stream output buffer to start at, or whether to append to the end of an existing buffer. The Offset must be DWORD aligned. If Stream Offset is equal to 0xFFFFFFFF then load the value at the Stream Output Buffer Offset address into SO_WRITE_OFFSET[Buffer#]. Otherwise, SO_WRITE_OFFSET[Buffer#] = Stream Offset.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>FFFFFFFFh</td><td></td><td>Load the value at the Stream Output Buffer Offset address into MMIO_SO_OFFSET[Buffer#].</td></tr>
<tr><td>xxxxxxxx xxxxxxxx
xxxxxxxx xxxxxx00b</td><td></td><td>MMIO_SO_OFFSET[Buffer#] = Stream Offset</td></tr>
</table></td></tr>
</table>

3DSTATE_SO_DECL_LIST

3DSTATE_SO_DECL_LIST

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 1h 3DSTATE_NONPIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 17h 3DSTATE_SO_DECL_LIST Format: OpCode
	15:9	**Reserved** Format: MBZ
	8:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: [1,257] / Excludes DWORD (0,1) 0-128 Entries / Value = 2 * (# of SO_DECL quads) + 1
1	31:16	**Reserved** Format: MBZ
	15:12	**Stream to Buffer Selects [3]** Format: U4 bitmask Identifies to which SO Buffers stream 3 outputs. See Stream To Buffer Selects [0] field description.
	11:8	**Stream to Buffer Selects [2]** Format: U4 bitmask Identifies to which SO Buffers stream 2 outputs. See Stream To Buffer Selects [0] field description.

3DSTATE_SO_DECL_LIST

	7:4	**Stream to Buffer Selects [1]** Format: U4 bitmask Identifies to which SO Buffers stream 1 outputs. See Stream To Buffer Selects [0] field description.
	3:0	**Stream to Buffer Selects [0]** Format: U4 bitmask Identifies to which SO Buffers stream 0 outputs (irrespective of whether those buffers are enabled via 3DSTATE_STREAMOUT). Software is required to scan the SO_DECL list in order to provide this summary information. Note: For "inactive" streams, software must program this field to all zero (no buffers written to) and the corresponding Num Entries field to zero (no valid SO_DECLs). Value: 1xxxb — Name: SO Buffer 3 Value: x1xxb — Name: SO Buffer 2 Value: xx1xb — Name: SO Buffer 1 Value: xxx1b — Name: SO Buffer 0
2	31:24	**Num Entries [3]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 3. (See notes in Num Entries [0] field description). Value: [0,128] — Name: entries
	23:16	**Num Entries [2]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 2. (See notes in Num Entries [0] field description). Value: [0,128] — Name: entries
	15:8	**Num Entries [1]** Format: U8 #entries Specifies the number of valid SO_DECL entries for Stream 1. (See notes in Num Entries [0] field description). Value: [0,128] — Name: entries

3DSTATE_SO_DECL_LIST

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

3DSTATE_STENCIL_BUFFER

3DSTATE_STENCIL_BUFFER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description	Project
This command sets the surface state of the separate stencil buffer, delivered as a pipelined state command. However, the state change pipelining isn't completely transparent (see restriction below).	
WM HW will internally manage the draining pipe and flushing of the caches when this command is issued. The PIPE_CONTROL restrictions are removed.	CHV, BSW

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

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

3DSTATE_STENCIL_BUFFER

Value	Name	Project
3h	Excludes Dword (0,1) **[Default]**	CHV, BSW

1	31	**Stencil Buffer Enable** Project: CHV, BSW Format: U1 When set indicates that there is a valid stencil buffer. **Programming Notes** This bit should be "0" if Depth buffer surface format is D16_UNORM OR Depth buffer surface type is NULL.
	30:29	**Reserved** Format: MBZ
	28:22	**Stencil Buffer Object Control State** Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for the stencil buffer.
	21:17	**Reserved** Format: MBZ
	16:0	**Surface Pitch** Format: U17-1 Pitch in Bytes This field specifies the pitch of the stencil buffer in (#Bytes - 1).

Value	Name	Description
[127, 1FFFFh]		corresponding to [128B, 128KB]also restricted to a multiple of 128B

Programming Notes

Since this surface is tiled, the pitch specified must be a multiple of the tile pitch, in the range [128B, 128KB].

The pitch must be set to 2x the value computed based on width, as the stencil buffer is stored with two rows interleaved. For details on the separate stencil buffer storage format in memory, see GPU Overview (vol1a), Memory Data Formats, Surface Layout, 2D Surfaces, Stencil Buffer Layout (section 8.20.4.8).

2..3 **Project:** CHV, BSW	63:0	**Surface Base Address** Project: CHV, BSW Format: GraphicsAddress[63:0]Stencil_Buffer This field specifies the address of the buffer in Graphics Memory. **Programming Notes** The Stencil Buffer can only be mapped to Main Memory (uncached).

3DSTATE_STENCIL_BUFFER

4 **Project:** CHV, BSW

31:15 **Reserved**

Project:	CHV, BSW
Format:	MBZ

14:0 **Surface QPitch**

Project:	CHV, BSW
Format:	QPitch[16:2]

Description	Project
This field specifies the distance in rows between array slices. It is used only in the following cases: • **Surface Array** is enabled *OR* • **Number of Mulitsamples** is not NUMSAMPLES_1 and **Multisampled Surface Storage Format** set to MSFMT_MSS *OR* • **Surface Type** is SURFTYPE_CUBE	CHV, BSW

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

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

3DSTATE_STREAMOUT

3DSTATE_STREAMOUT

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command contains pipelined state required by the SOL unit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Eh 3DSTATE_STREAMOUT Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 3h Excludes DWord (0,1) Project: CHV, BSW Format: =n Total Length - 2
1	31	**SO Function Enable** Project: All Format: U1 If set, the SO function is enabled. Vertex data will be streamed out to memory (subject to overflow detection) as controlled by the various SO-related state variables. If clear, the SO function is disabled, and therefore no vertex data will be streamed out to memory. However, the Rendering Disable and Render Stream Select fields will still be used to determine which vertices (if any) are forwarded down the pipeline for (possible) rendering.

3DSTATE_STREAMOUT

30 API Rendering Disable

Project:	CHV, BSW
Format:	U1

If set, Indicates the API wants the SO stage not to forward any topologies down the pipeline. If clear, Indicates the API wants the SO stage to forward topologies associated with **Render Stream Select** down the pipeline. This bit is used even if **SO Function Enable** is DISABLED.

Programming Notes
The SOL unit generates an SOL_INT::Render_Enable which ultimately controls whether rendering occurs or not.

29 Reserved

Project:	All
Format:	MBZ

28:27 Render Stream Select

Project:	All
Format:	U2

Description
This field specifies which stream has been selected to be forwarded down the pipeline for possible rendering. Topologies from other streams will not be passed down the pipeline. If Rendering Disable is set, this field is ignored, as no topologies are sent down the pipeline.
SO Function Enable must also be ENABLED in order for thiis field to select a stream for rendering. When **SO Function Enable** is DISABLED and Rendering Disable is cleared (i.e., rendering is enabled), StreamID is ignored downstream of the SO stage, allowing any stream to be rendered.

26 Reorder Mode

Project:	All

This bit controls how vertices of triangle objects in TRISTRIP[_ADJ] and TRISTRIP_REV are reordered for the purposes of stream-out only (does not impact rendering). See table in Input Buffering.

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

25 SO Statistics Enable

Project:	All
Format:	Enable

This bit controls whether StreamOutput statistics register(s) can be incremented.

3DSTATE_STREAMOUT

Value	Name	Description	Project
0h	Disable	SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers cannot increment.	All
1h	Enable	SO_NUM_PRIMS_WRITTEN[0..3] and SO_PRIM_STORAGE_NEEDED[0..3] registers can increment.	All

24:23 **Force Rendering**

Project:	CHV, BSW

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

Value	Name	Description	Project
0h	Normal	SOL_INT::Render_Enable is computed normally	All
1h	Resreved		All
2h	Force_Off	Forces the rendering to be disabled.	All
3h	Force_on	Forces the rendering to be enabled.	All

22:12 **Reserved**

Project:	All
Format:	MBZ

11:8 **Reserved**

Project:	CHV, BSW
Format:	MBZ

7:0 **Reserved**

Project:	All
Format:	MBZ

2

31:30 **Reserved**

Project:	All
Format:	MBZ

29 **Stream 3 Vertex Read Offset**

Project:	All
Format:	U1 count of 256-bit units

Specifies amount of data to skip over before reading back Stream 3 vertex data. (See **Stream 0 Vertex Read Offset**)

28:24 **Stream 3 Vertex Read Length**

Project:	All
Format:	U5-1 count of 256-bit units

(See Stream 0 Vertex Read Length)

23:22 **Reserved**

3DSTATE_STREAMOUT

	Project:	All
	Format:	MBZ

21 Stream 2 Vertex Read Offset

Project:	All
Format:	U1 count of 256-bit units

Specifies amount of data to skip over before reading back Stream 2 vertex data. (See Stream 0 Vertex Read Offset)

20:16 Stream 2 Vertex Read Length

Project:	All
Format:	U5-1 count of 256-bit units

15:14 Reserved

Project:	All
Format:	MBZ

13 Stream 1 Vertex Read Offset

Project:	All
Format:	U1 count of 256-bit units

Specifies amount of data to skip over before reading back Stream 1 vertex data. (See Stream 0 Vertex Read Offset)

12:8 Stream 1 Vertex Read Length

Project:	All
Format:	U5-1 count of 256-bit units

(See Stream 0 Vertex Read Length)

7:6 Reserved

Project:	All
Format:	MBZ

5 Stream 0 Vertex Read Offset

Project:	All
Format:	U1 count of 256-bit units

Specifies amount of data to skip over before reading back Stream 0 vertex data. Must be zero if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).

4:0 Stream 0 Vertex Read Length

Project:	All
Format:	U5-1 count of 256-bit units

3DSTATE_STREAMOUT

		Specifies amount of vertex data to read back for Stream 0 vertices, starting at the Stream 0 Vertex Read Offset location. Maximum readback is 17 256-bit units (34 128-bit vertex attributes). Read data past the end of the valid vertex data has undefined contents, and therefore shouldn't be used to source stream out data. Must be zero (i.e., read length = 256b) if the GS is enabled and the Output Vertex Size field in 3DSTATE_GS is programmed to 0 (i.e., one 16B unit).
3 **Project:** CHV, BSW	31:28	**Reserved** Project: CHV, BSW Format: MBZ
	27:16	**Buffer 1 Surface Pitch** Project: CHV, BSW
	15:12	**Reserved** Project: CHV, BSW Format: MBZ
	11:0	**Buffer 0 Surface Pitch** Project: CHV, BSW Format: U12 pitch in Bytes This field specifies the pitch of the SO buffer in #Bytes. **Value** [0,2048] — **Name** Must be 0 or a multiple of 4 Bytes. **Programming Notes** A Surface Pitch of 0 indicates an un-bound buffer. No writes are performed. Surface Base Address is ignored.
4 **Project:** CHV, BSW	31:28	**Reserved** Project: CHV, BSW Format: MBZ
	27:16	**Buffer 3 Surface Pitch** Project: CHV, BSW Format: U12
	15:12	**Reserved** Project: CHV, BSW Format: MBZ
	11:0	**Buffer 2 Surface Pitch** Project: CHV, BSW Format: U12

3DSTATE_TE

3DSTATE_TE		
Project:	CHV, BSW	
Source:	RenderCS	
Length Bias:	2	
For [CHV, BSW], the state used by TE is defined with this inline state packet.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 1Ch 3DSTATE_TE Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:20	**Reserved** Project: All Format: MBZ
	19	**Reserved** Project: CHV, BSW Format: MBZ
	18:16	**Reserved** Project: CHV, BSW Format: MBZ

3DSTATE_TE

15:14 Reserved

Project:	All
Format:	MBZ

13:12 Partitioning

Project:	All
Format:	U2

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

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

11:10 Reserved

Project:	All
Format:	MBZ

9:8 Output Topology

Project:	All
Format:	U2

This field specifies which primitive types are to be output.

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

7:6 Reserved

Project:	All
Format:	MBZ

5:4 TE Domain

Project:	All
Format:	U2

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

3DSTATE_TE

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

3 **Reserved**

Project:	All
Format:	MBZ

2:1 **TE Mode**

Project:	All
Format:	U2

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

Value	Name	Description	Project
0h	HW_TESS	Normal HW Tessellation Mode. The TessFactors are read from the patch URB entry, and are used to perform fixed-function hardware tessellation of the specified domain.	All
1h	SW_TESS	Software Tessellation Mode. The TE unit will pass down HS-thread-generated tessellated domain points instead of generating them itself from TessFactors. The TE unit will read the Domain Point Count and Domain Point Buffer Starting Address fields from the patch header, and if the count is 0 it will consider the patch culled and discard it. Otherwise the address is used to start fetching DOMAIN_POINT structures from memory and passing them down the pipeline to DS.	CHV, BSW

0 **TE Enable**

Project:	All
Format:	Enable

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

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

2 31:0 **Maximum Tessellation Factor Odd**

Project:	All
Format:	IEEE_Float

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

3DSTATE_TE

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

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

3 | 31:0 **Maximum Tessellation Factor Not Odd**

Project:	All
Format:	IEEE_Float

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

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

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

3DSTATE_URB_DS

3DSTATE_URB_DS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description

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

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 32h 3DSTATE_URB_DS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All

3DSTATE_URB_DS

		Format: =n
1	31:25	**DS URB Starting Address** Project: CHV, BSW Format: U7 Offset from the start of the URB memory where DS starts its allocation, specified in multiples of 8 KB.

Value	Name	Project	Exists If
[0,48]		CHV, BSW	Device[SliceCount] == 1
[4,48]		CHV, BSW	Device[SliceCount] GT 1

	24:16	**DS URB Entry Allocation Size** Project: All Format: U9-1 Count of 512-bit units Specifies the length of each URB entry owned by DS. This field is always used (even if DS Function Enable is DISABLED).

Value	Name
[0,9]	

	15:0	**DS Number of URB Entries** Project: All

Description
Specifies the number of URB entries that are used by DS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below. This field is always used (even if DS Function Enable is DISABLED).
If Domain Shader Thread Dispatch is Enabled then the minimum number of handles that must be allocated is 34 URB entries.

Value	Name
[0,1536]	
[0,384]	

Programming Notes
DS Number of URB Entries must be divisible by 8 if the DS URB Entry Allocation Size is programmed to a value less than 9, which is 10 512-bit URB entries. "2:0" = reserved "000"

3DSTATE_URB_GS

3DSTATE_URB_GS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description

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

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 33h 3DSTATE_URB_GS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All

3DSTATE_URB_GS

		Format: =n
1	31:25	**GS URB Starting Address** Project: CHV, BSW Format: U7 Offset from the start of the URB memory where GS starts its allocation, specified in multiples of 8 KB.

Value	Name	Project	Exists If
[0,48]			Device[SliceCount] == 1
[4,48]			Device[SliceCount] GT 1

24:16	**GS URB Entry Allocation Size** Project: All Format: U9-1 512-bit units Specifies the length of each URB entry owned by GS. This field is always used (even if GS Function Enable is DISABLED).
15:0	**GS Number of URB Entries** Project: All Specifies the number of URB entries that are used by GS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below. This field is always used (even if GS Function Enable is DISABLED).

Value	Name	Project
[0,960]		CHV, BSW
[0,256]		CHV, BSW

Programming Notes
Only if GS is disabled can this field be programmed to 0. If GS is enabled this field shall be programmed to a value greater than 0. For GS Dispatch Mode "Single", this field shall be programmed to a value greater than or equal to 1. For other GS Dispatch Modes, refer to the definition of Dispatch Mode (3DSTATE_GS) for minimum values of this field.
GS Number of URB Entries must be divisible by 8 if the GS URB Entry Allocation Size is less than 9 512-bit URB entries. "2:0" = reserved "000"

3DSTATE_URB_HS

3DSTATE_URB_HS

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description

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

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

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 31h 3DSTATE_URB_HS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n

3DSTATE_URB_HS

1	31:25	**HS URB Starting Address** Project: CHV, BSW Format: U7 Offset from the start of the URB memory where HS starts its allocation, specified in multiples of 8 KB.

Value	Name	Project	Exists If
[0,48]			Device[SliceCount] == 1
[4,48]			Device[SliceCount] GT 1

	24:16	**HS URB Entry Allocation Size** Project: All Format: U9-1 Count of 512-bit units Specifies the length of each URB entry owned by HS. This field is always used (even if HS Function Enable is DISABLED).
	15:0	**HS Number of URB Entries** Project: All Specifies the number of URB entries that are used by HS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below. This field is always used (even if HS Function Enable is DISABLED). Programming Restriction:HS Number of URB Entries must be divisible by 8 if the HS URB Entry Allocation Size is less than 9 512-bit URB entries."2:0" = reserved "000"

Value	Name	Project
[0,504]		CHV, BSW
[0,80]		CHV, BSW

3DSTATE_URB_VS

3DSTATE_URB_VS

Project: CHV, BSW

Source: RenderCS, PositionCS

Length Bias: 2

Description

VS URB Entry Allocation Size equal to 4(5 512-bit URB rows) may cause performance to decrease due to banking in the URB. Element sizes of 16 to 20 should be programmed with six 512-bit URB rows.

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

The offset and size should be programmed as if there is only one slice enabled. Hardware will grow the size based on the slice configuration. Software shall ensure that the values programmed do not exceed the URB capacity of one slice. Refer to the L3 allocation and programming guide for valid URB configurations.

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 30h 3DSTATE_URB_VS Format: OpCode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n

3DSTATE_URB_VS

1 — 31:25 — VS URB Starting Address

Project:	CHV, BSW
Format:	U7

Offset from the start of the URB memory where VS starts its allocation, specified in multiples of 8 KB.

Value	Name	Project	Exists If
[0,48]			Device[SliceCount] == 1
[4,48]			Device[SliceCount] GT 1

24:16 — VS URB Entry Allocation Size

Project:	All
Format:	U9-1 count of 512-bit units

Specifies the length of each URB entry owned by VS. This field is always used (even if VS Function Enable is DISABLED).

Value	Name	Project
[0,9]		CHV, BSW

Programming Notes
Programming Restriction: As the VS URB entry serves as both the per-vertex input and output of the VS shader, the VS URB Allocation Size must be sized to the maximum of the vertex input and output structures.

15:0 — VS Number of URB Entries

Project:	All
Format:	U16

Specifies the number of URB entries that are used by VS, based on only 1 slice enabled. When multiple slices are enabled, HW will multiply the value programmed by the number of slices in order to determine the total number of entries. SW shall ensure that the total number of entries does not exceed the relevant ValidValue range listed below.
This field is always used (even if VS Function Enable is DISABLED).

Value	Name
[64,2560]	
[34,640]	

Programming Notes
Programming Restriction: VS Number of URB Entries must be divisible by 8 if the VS URB Entry Allocation Size is less than 9 512-bit URB entries."2:0" = reserved "000b"
When tessellation is enabled, the VS Number of URB Entries must be greater than or equal to 192.

3DSTATE_VERTEX_BUFFERS

3DSTATE_VERTEX_BUFFERS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

Description
This command is used to specify VB state used by the VF function.
[CHV, BSW]: Can specify from 1 to 33 VBs.
The VertexBufferID field within a VERTEX_BUFFER_STATE structure indicates the specific VB. If a VB definition is not included in this command, its associated state is left unchanged and is available for use if previously defined.

Programming Notes
It is possible to have individual vertex elements sourced completely from generated ID values and therefore not require any vertex buffer accesses for that vertex element. In this case, VF function will simply ignore the VB state associated with that vertex element. If all enabled vertex elements have this characteristic, no VBs are required to process 3DPRIMITIVE commands. For example, this might arise when the user wants to perform all data lookups in the first shader, so only generated index values need to be passed down to it. In this extreme case, SW would not need to program any VB state, and therefore not need to issue any 3DSTATE_VERTEX_BUFFERS commands.
For any 3DSTATE_VERTEX_BUFFERS command, at least one VERTEX_BUFFER_STATE structure must be included.
VERTEX_BUFFER_STATE structures are 4 DWords for both VERTEXDATA buffers and INSTANCEDATA buffers.
Inclusion of partial VERTEX_BUFFER_STATE structures is UNDEFINED.
The order in which VBs are defined within this command can be arbitrary, though a vertex buffer must be defined only once in any given command (otherwise operation is UNDEFINED).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h 3D Format: Opcode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_VERTEX_BUFFERS Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 08h 3DSTATE_VERTEX_BUFFERS Format: Opcode

3DSTATE_VERTEX_BUFFERS

	15:8	**Reserved**
	7:0	**DWord Length** Format: =n n = 4b-1 (where b = # of buffer states included) **Value** / **Name** / **Project** 3 / DWORD_COUNT_n **[Default]** / [3,131] / 1-33 Buffers / CHV, BSW
1..n	127:0	**Vertex Buffer State** Format: VERTEX_BUFFER_STATE [CHV, BSW]

3DSTATE_VERTEX_ELEMENTS

3DSTATE_VERTEX_ELEMENTS

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

Description
This is a variable-length command used to specify the active vertex elements. Each VERTEX_ELEMENT_STATE structure contains a Valid bit which determines which elements are used.
[CHV, BSW]: Up to 34 elements.

Programming Notes
[CHV, BSW]: At least one VERTEX_ELEMENT_STATE structure must be included.
The 3DSTATE_VERTEX_ELEMENTS must not be programmed more than once before each 3DPRIMITIVE command.
Inclusion of partial VERTEX_ELEMENT_STATE structures is UNDEFINED.
[CHV, BSW]: SW must ensure that at least one vertex element is defined prior to issuing a 3DPRIMTIVE command, or operation is UNDEFINED.
[CHV, BSW]: There are no 'holes' allowed in the destination vertex: NOSTORE components must be overwritten by subsequent components unless they are the trailing DWords of the vertex. Software must explicitly chose some value (probably 0) to be written into DWords that would otherwise be 'holes'.
[CHV, BSW]: Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.
[CHV, BSW]: See additional restrictions listed in the command fields and VERTEX_ELEMENT_STATE description.
[CHV, BSW]: Element[0] must be valid.
[CHV, BSW]: All elements must be valid from Element[0] to the last valid element. (I.e. if Element[2] is valid then Element[1] and Element[0] must also be valid).
[CHV, BSW]: The pitch between elements packed in the URB will always be 128 bits.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 3h 3D Format: Opcode

3DSTATE_VERTEX_ELEMENTS

	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_VERTEX_ELEMENTS Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 09h 3DSTATE_VERTEX_ELEMENTS Format: Opcode
	15:8	**Reserved**
	7:0	**DWord Length** Format: =n Vertex Element Count = (DWord Count + 1) / 2
1..n	63:0	**Element** Format: VERTEX_ELEMENT_STATE

Value	Name	Description	Project
1	DWORD_COUNT_n **[Default]**	excludes DWords 0,1	
[1,67]	Range	1-34 Elements	CHV, BSW

3DSTATE_VF

3DSTATE_VF

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command is used to set various state variables in the VF stage.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0Ch 3DSTATE_VF Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11	**Reserved**
	10	**Reserved**
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8	**Indexed Draw Cut Index Enable** Project: All Format: Enable If ENABLED, vertex indices in RANDOM 3DPRIMITIVE commands are compared to the Cut Index (specified below). When the vertex index matches the Cut Index any previous topology is terminated.If DISABLED, vertex indices are not compared to the Cut Index and are used strictly as indices into vertex buffers.This field can only be enabled for certain primitive topology types. Refer to the table later in this section for details.

3DSTATE_VF

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:0	**Cut Index** Project: All This field specifies the index value that is considered the "cut index" which vertex indices are compared to if a Cut Index Enable is set. The Cut Index is compared to the fetched (and possibly-sign-extended) vertex index, and if these values are equal, the current primitive topology is terminated. Note that, for index buffers <32bpp, it is possible to set the Cut Index to a (large) value that will never match a sign-extended vertex index.

3DSTATE_VF_INSTANCING

3DSTATE_VF_INSTANCING

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command is used to control the "instancing" state associated with a specific vertex element.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 49h 3DSTATE_VF_INSTANCING Format: OpCode
	15:8	**Reserved**
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 1h — Excludes DWord (0,1) **[Default]** Value: 43h — Context Restore
1	31:9	**Reserved**
	8	**Instancing Enable** Format: Enable Value: 0h — Name: Disabled — Description: This vertex element is not instanced and therefore vertices within instances can each receive different data for this vertex element. Within each instance, the source vertex data for this vertex element is determined according the the Vertex Access Type of the 3DPRIMITIVE command. Instance Data Step Rate is ignored for this vertex element.

3DSTATE_VF_INSTANCING

		1h	Enabled	This vertex element is instanced and therefore vertices within instances will receive the same data for this vertex element. The source pointer for this particular vertex element will be (a) initialized at the start of 3DPRIMITIVE processing, (b) held constant for all vertices within an instance, and (c) advanced between instances as a function of Instance Data Step Rate.
	7:6	**Reserved** Format: MBZ		
	5:0	**Vertex Element Index** Format: U6 This field identifies which vertex element state is to be modified by this command. **Value** / **Name** [0,33]		
2	31:0	**Instance Data Step Rate** If Instancing Enable is ENABLED, this field determines the rate at which data for this particular vertex element is changed between instances. Only after the number of instances specified by this field is generated is new (sequential) vertex element data provided. This process continues for each group of instances defined in the 3DPRIMTIVE command. For example, a value of 1 in this field causes new data to be supplied for this vertex element with each sequential (instance) group of vertices. A value of 2 causes every other instance group of vertices to be provided with new vertex element data. The special value of 0 causes all vertices of all instances generated by the 3DPRIMITIVE command to be provided with the same data for this vertex element. (The same effect can be achieved by setting this field to its maximum value.) If Instancing Enable is DISABLED, this field is ignored.		

3DSTATE_VF_SGVS

3DSTATE_VF_SGVS

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

This command is used to control the insertion of the VertexID and InstanceID System-Generated Values (SGVs) into an input Vertex URB Entry (VUE) (available as input to a VS thread). VertexID and InstanceID insertion can be individually controlled. The insertion locations are specified as 128-bit element locations (starting at the beginning of the VUE) and the 32-bit component within those specified elements. The SGV values can be inserted either (a) within a valid vertex element (in which case the value overwrites the value specified via 3DSTATE_VERTEX_ELEMENTS) or (b) beyond the last valid vertex element written to the URB. This permits some orthogonality between the programming of vertex elements (which typically is known at draw time) and programming of SGV insertion (which is associated with the shader). There are some restrictions however (see below). If an SGV is inserted beyond the last valid vertex element, zeroes are first inserted in the VUE after the last valid vertex element up to and including the vertex element receiving an SGV. If both of the SGVs are enabled for insertion, the zeroes will extend to the last (largest index) vertex element receiving an SGV. Then the SGV(s) are inserted.

Programming Notes

Programming Restrictions:

- It is INVALID to store both the VertexID and InstanceID in the same element/component location within the VUE.
- The states programmed by this command overwrite the state programmed by any previous commands. I.e., VertexID and InstanceID (if enabled) can only be inserted in one component of a vertex.
- It is INVALID to insert an SGV value past the end of the VUE entry (as determined by VS URB Entry Allocation Size) or past the 33rd vertex element. Therefore the programming of VS URB Entry Allocation Size needs to comprehend any SGV insertion requirements.
- It is INVALID to use this command to overwrite any portion of a 64-bit vertex element component.
- It is INVALID to use this command to overwrite a EdgeFlag vertex element component or any vertex element beyond it.

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

3DSTATE_VF_SGVS

23:16 3D Command Sub Opcode

Default Value:	4Ah 3DSTATE_VF_SGVS
Format:	OpCode

15:8 Reserved

Project:	All
Format:	MBZ

7:0 DWord Length

Default Value:	0h Excludes DWord (0,1)
Project:	All
Format:	=n Total Length - 2

1 31 InstanceID Enable

Project:	All
Format:	Boolean

Value	Name	Description	Project
0h	Disabled	InstanceID is not inserted	All
1h	Enabled	InstanceID is inserted	All

30:29 InstanceID Component Number

Project:	All

If InstanceID Enable is ENABLED, this field specifies the 32-bit component location (within the 4-component VUE) where it is inserted.
If InstanceID Enable is DISABLED, this field is ignored.

Value	Name	Description	Project
0	COMP_0	If enabled, InstanceID is inserted in component 0 (.x)	All
1	COMP_1	If enabled, InstanceID is inserted in component 1 (.y)	All
2	COMP_2	If enabled, InstanceID is inserted in component 2 (.z)	All
3	COMP_3	If enabled, InstanceID is inserted in component 3 (.w)	All

28:22 Reserved

21:16 InstanceID Element Offset

Project:	All
Format:	U6 Offset of 128-bit element

If InstanceID Enable is ENABLED, this field specifies the VUE element offset of the 128-bit element where it is to be inserted. The InstanceID Component Number specifies where in the specified element it is inserted.

Value	Name
[0,33]	

3DSTATE_VF_SGVS

15 **VertexID Enable**

Project:	All
Format:	Boolean

Value	Name	Description	Project
0h	Disabled	VertexID is not inserted	All
1h	Enabled	VertexID is inserted	All

14:13 **VertexID Component Number**

Project:	All

If VertexID Enable is ENABLED, this field specifies the 32-bit component location (within the 4-component VUE) where it is inserted.If VertexID Enable is DISABLED, this field is ignored.

Value	Name	Description	Project
0	COMP_0	If enabled, VertexID is inserted in component 0 (.x)	All
1	COMP_1	If enabled, VertexID is inserted in component 1 (.y)	All
2	COMP_2	If enabled, VertexID is inserted in component 2 (.z)	All
3	COMP_3	If enabled, VertexID is inserted in component 3 (.w)	All

12:6 **Reserved**

Project:	All
Format:	MBZ

5:0 **VertexID Element Offset**

Project:	All
Format:	U6 Offset of 128-bit element

If VertexID Enable is ENABLED, this field specifies the VUE element offset of the 128-bit element where it is to be inserted. The VertexID Component Number specifies where in the specified element it is inserted. This is also the vertex element index. If VertexID Enable is DISABLED, this field is ignored.

Value	Name
[0,33]	

3DSTATE_VF_STATISTICS

3DSTATE_VF_STATISTICS

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

The VF stage tracks two pipeline statistics, the number of vertices fetched and the number of objects generated. VF will increment the appropriate counter for each when statistics gathering is enabled by issuing the 3DSTATE_VF_STATISTICS command with the [Statistics Enable] bit set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Format: Opcode Value 1h: Pipelined, Single DWord **[Default]** — Project: CHV, BSW
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	23:16	**3D Command Sub Opcode** Default Value: 0Bh 3DSTATE_VF_STATISTICS Format: Opcode GFXPIPE[28:27 = 1h, 26:24 = 0h, 23:16 = 0Bh] (Pipelined, Single DWord)
	15:1	**Reserved** Format: MBZ
	0	**Statistics Enable** Format: Enable If ENABLED, VF will increment the pipeline statistics counters IA_VERTICES_COUNT and IA_PRIMITIVES_COUNT for each vertex fetched and each object output, respectively, for 3DPRIMITIVE commands issued subsequently. If DISABLED, these counters will not be incremented for subsequent 3DPRIMITIVE commands.

3DSTATE_VF_TOPOLOGY

3DSTATE_VF_TOPOLOGY

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command specifies the VF stage's Topology state which can be used to override the Primitive Topology Type in subsequent 3DPRIMITIVE commands.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Bh 3DSTATE_VF_TOPOLOGY Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:6	**Reserved** Project: All Format: MBZ
	5:0	**Primitive Topology Type** Project: All Format: 3D_Prim_Topo_Type [CHV, BSW] This field specifies the VF stage's Topology state.

3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_CC

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CC command is used to define the location of fixed functions' viewport state table.

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

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The 3DSTATE_VIEWPORT_STATE_POINTERS_CLIP command is used to define the location of fixed functions' viewport state table.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 21h 3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Format: =n
1	31:6	**SF Clip Viewport Pointer** Project: All Format: DynamicStateOffset[31:6]SF_CLIP_VIEWPORT*16 Specifies the 64-byte aligned address offset of the SF_CLIP_VIEWPORT state. This offset is relative to the Dynamic State Base Address.
	5:0	**Reserved** Project: Format: MBZ

3DSTATE_VS

3DSTATE_VS

Project: CHV, BSW
Source: RenderCS, PositionCS
Length Bias: 2

Description	Project
This command specifies most of the state used by the Vertex Shader (VS) stage.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 10h 3DSTATE_VS Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 7h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1..2	63:6	**Kernel Start Pointer** Project: All Format: InstructionBaseOffset[63:6]Kernel This field specifies the starting location of the kernel program run by threads spawned by the VS pipeline stage. It is specified as a 64-byte-granular offset from the Instruction Base Address. This field is ignored if VS Function Enable is DISABLED.
	5:0	**Reserved**

3DSTATE_VS

3 / 31 — Single Vertex Dispatch

Project:	
Format:	U1 Enumerated Type

When this bit is set, SIMD4x2 VS threads will only process a single vertex, otherwise SIMD4x2 threads will process either one or two vertices. This field is ignored if **SIMD8 Dispatch Enable** is set.

Value	Name	Description	Project
0h	Multiple	Dual vertex SIMD4x2 thread dispatches are allowed.	All
1h	Single	Single vertex SIMD4x2 thread dispatches are forced.	All

30 — Vector Mask Enable

Project:	All

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

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

Programming Notes

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

29:27 — Sampler Count

Project:	All
Format:	U3

This field specifies (in multiples of 4) the number of sets of sampler state that will be prefetched for use by the VS kernel. While the prefetching of sampler state is optional and does not impact functionality, it may improve performance.
This field is ignored if the Function Enable state is set to DISABLED.

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

26 — Reserved

3DSTATE_VS

Project:	All
Format:	MBZ

25:18 **Binding Table Entry Count**

Project:	All
Format:	U8

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

Value	Name
[0,255]	

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

17 **Thread Dispatch Priority**

Project:	All
Format:	U1 Enumerated Type

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

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

16 **Floating Point Mode**

Project:	All
Format:	U1 Enumerated Type

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

Value	Name	Description	Project
0h	IEEE-754	Use IEEE-754 Rules	All
1h	Alternate	Use Alternate Rules	All

3DSTATE_VS

15:14 Reserved

Project:	All
Format:	MBZ

13 Illegal Opcode Exception Enable

Project:	All
Format:	Enable

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

12 Accesses UAV

Format:	Enable

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

Programming Notes	Project
This field must not be set when VS Function Enable is disabled.	
Workaround: If the vertex shader is the last shader to have UAV access, a PIPE_CONTROL with **CS_STALL** must be sent before the 3dprimitive using the UAV access.	CHV, BSW

11:8 Reserved

7 Software Exception Enable

Project:	All
Format:	Enable

This bit gets loaded into EU CR0.1[13] (note the bit # difference). See Exceptions and ISA Execution Environment. This field is ignored if VS Function Enable is DISABLED.

6:0 Reserved

Project:	All
Format:	MBZ

4..5 63:10 Scratch Space Base Pointer

Project:	All
Format:	GeneralStateOffset[63:10]ScratchSpace

Specifies the starting location of the scratch space area allocated to this FF unit as a 1K-byte aligned offset from the General State Base Address. If required, each thread spawned by this FF unit will be allocated some portion of this space, as specified by Per-Thread Scratch Space. The computed offset of the thread-specific portion will be passed in the thread payload as Scratch Space Offset. The thread is expected to utilize "stateless" DataPort read/write requests to access scratch space, where the DataPort will cause the General State Base Address to be added to the offset passed in the request header. This field is ignored if VS Function Enable is DISABLED. In 64b OS all pointers need to be seen by SW as 48b. HW does not support a Scratch Space Base Pointer larger than 32b, therefore SW must ensure the scratch space base offset upper is set to 0's. But the DWORD must be in the command for SW to right the pointer to.

3DSTATE_VS

	9:4	**Reserved** Project: All Format: MBZ
	3:0	**Per-Thread Scratch Space** Project: All Format: U4 power of 2 Bytes over 1K Bytes Specifies the amount of scratch space to be allocated to each thread spawned by this FF unit. The driver must allocate enough contiguous scratch space, starting at the Scratch Space Base Pointer, to ensure that the Maximum Number of Threads can each get Per-Thread Scratch Space size without exceeding the driver-allocated scratch space. This field is ignored if VS Function Enable is DISABLED. Value: [0,11] — Name: — Description: Indicating [1K Bytes, 2M Bytes] **Programming Notes** This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it.
6	31:25	**Reserved** Format: MBZ
	24:20	**Dispatch GRF Start Register For URB Data** Project: All Format: U5 Specifies the starting GRF number for the URB portion (URB constants and vertices) of the thread payload. This field is ignored if VS Function Enable is DISABLED. Value: [0,31] — Name: — Description: indicating GRF [R0, R31]
	19:17	**Reserved** Project: All Format: MBZ
	16:11	**Vertex URB Entry Read Length** Format: U6 Specifies the number of pairs of 128-bit vertex elements to be passed into the payload for each vertex. This field is ignored if VS Function Enable is DISABLED. For SIMD4x2 dispatch, each vertex element requires one GRF of payload data, therefore the number of GRFs with vertex data will be double the value programmed in this field. For SIMD8 dispatch, each vertex element requires 4 GRFs of payload data, therefore the number of GRFs with vertex data will be 8 times the value programmed in this field. The EU limit of 128 GRFs imposes a maximum limit of 30 elements per vertex pushed into the payload, though the practical limit may be lower. If input vertices exceed

3DSTATE_VS

the practical limit, software must decide between resorting to pulling elements during thread execution or dropping back to SIMD4x2 dispatch. Note that the VUE is used for both input and output, so when using the pull-model software must ensure inputs are not overwritten before last use.

Value	Name	Description
[1,63]		if SIMD8 dispatch disabled
[0,15]		if SIMD8 dispatch enabled

10 Reserved

Project:	All
Format:	MBZ

9:4 Vertex URB Entry Read Offset

Project:	All
Format:	U6

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

Value	Name
[0,63]	

3:0 Reserved

Project:	All
Format:	MBZ

7 31:23 Maximum Number of Threads

Project:	CHV, BSW
Format:	U9-1 Thread count

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

Value	Name	Description	Project
[0,503]		indicating thread count of [1,504]	CHV, BSW
[0,79]		indicating thread count of [1,80]	CHV, BSW

22 Reserved

Project:	CHV, BSW
Format:	MBZ

21:11 Reserved

3DSTATE_VS

10 Statistics Enable

Project:	All
Format:	Enable

If ENABLED, the VS stage will perform statistics gathering. See the Statistics Gathering subsection.
If DISABLED, statistics information associated with the VS stage will be left unchanged.

9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:3 Reserved

Project:	All
Format:	MBZ

2 SIMD8 Dispatch Enable

Project:	All
Format:	Enable

This field determines how VS threads are dispatched and how the thread payloads are generated. The setting of this field must agree with how the VS kernel was compiled.
If ENABLED, SIMD8 VS thread dispatches are performed. The **Single Vertex Dispatch** field is ignored.
If DISABLED, SIMD4x2 thread dispatches are performed. The **Single Vertex Dispatch** field can be used to force single-vertex dispatches.

1 Vertex Cache Disable

Project:	All
Format:	Disable

This bit controls the operation of the Vertex Cache. This field is always used.
If the Vertex Cache is DISABLED and the VS Function is ENABLED, the Vertex Cache is not used and all incoming vertices will be passed to VS threads.
If the Vertex Cache is ENABLED and the VS Function is ENABLED, only incoming vertices that do not hit in the Vertex Cache will be passed to VS threads.
If the Vertex Cache is ENABLED and the VS Function is DISABLED, input vertices that miss in the Vertex Cache will be assembled and written to the URB (by the VF stage), and subsequently passed through the VS stage unmodified (i.e, no VS threads are spawned).
The Vertex Cache is invalidated whenever the Vertex Cache becomes DISABLED, whenever the VS Function Enable toggles, between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command.

Programming Notes
See the Vertex Caching subsection for details on implicit Vertex Cache disabling.

3DSTATE_VS

0 — Function Enable

Format:	Enable

This bit determines whether or not the VS stage spawns VS threads, which comprises the bulk of the VS stage functionality.
If ENABLED, VS threads may be spawned to process VF-generated vertices before the resulting vertices are passed down the pipeline.
If DISABLED, VF-generated vertices will pass thru the VS function and are sent down the pipeline unmodified. The Vertex Cache (if enabled) is still available.

8 — 31:28 — Reserved

Project:	All
Format:	MBZ

27 — Reserved

Project:	
Format:	MBZ

26:21 — Vertex URB Entry Output Read Offset

Project:	All
Format:	U6

Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB by the Setup Back-End (SBE) function. The offset programmed will specify the start of Attribute 0 to be passed in subsequent Pixel Shader thread payloads. Refer to the Attribute Interpolator Setup documentation.

Value	Name
[0,63]	

Programming Notes

As the vertex header data located at the start of the Vertex URB entry is typically only used by 3D pipeline FFs (i.e., Clipper, Setup FrontEnd) and not required as interpolated attributes in PIxel Shader threads, it is expected that SW will program this Start Offset skip over the vertex header.
This offset value is ignored if SBE's Number of SF Attributes state is programmed to 0 (i.e., no attributes are defined beyond the position read from the Vertex Header)

20:16 — Vertex URB Entry Output Length

Project:	All
Format:	U5

Specifies the amount of Vertex Attribute URB data to be read by the Setup Back-End function for each Vertex URB entry, in 256-bit units. The attribute data will be read starting at the offset specified by the Vertex URB Entry Output Read Offset state.

Value	Name
[1,16]	

3DSTATE_VS

		Programming Notes This length value is ignored if SBE's Number of SF Attributes state is programmed to 0 (i.e., no attributes are defined beyond the position read from the Vertex Header).
	15:8	**User Clip Distance Clip Test Enable Bitmask** Project: All Format: mask[8] This 8 bit mask field selects which of the 8 Clip Distance Values (if any) are to be included in the Clip stage's trivial reject / trivial accept / must clip determination function. The ClipDistance Values (if present) are located in DW8-15 of the VUE Vertex Header located at the beginning of VUE URB entries. Bit 0 of this field corresponds to Clip Distance Value 0.
	7:0	**User Clip Distance Cull Test Enable Bitmask** Project: All Format: mask[8] This 8 bit mask field selects which of the 8 Clip Distance Values (if any) are to be included in the Clip stage's trivial reject / trivial accept determination function. Note that must clip determination is not included in this function. The ClipDistance Values (if present) are located in DW8-15 of the VUE Vertex Header located at the beginning of VUE URB entries. Bit 0 of this field corresponds to Clip Distance Value 0.

3DSTATE_WM_CHROMAKEY

3DSTATE_WM_CHROMAKEY

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Ch 3DSTATE_WM__CHROMAKEY Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Default Value: 0h Excludes Dword (0,1) Project: All Format: =n Total Length - 2
1	31	**ChromaKey Kill Enable** Project: All Format: Enable If ENABLED, indicates that at least one of the attached samplers has ChromaKeyKill enabled.
	30:0	**Reserved** Project: All Format: MBZ

3DSTATE_WM_DEPTH_STENCIL

3DSTATE_WM_DEPTH_STENCIL

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command replaces the indirect state DEPTH_STENCIL_STATE with an inline state command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 4Eh 3DSTATE_WM_DEPTH_STENCIL Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Project: All Format: =n Total Length - 2 Value: 01h — Name: Excludes Dword (0,1) **[Default]** — Project: CHV, BSW
1	31:29	**Stencil Fail Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW] This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test fails. **Programming Notes** if all three stencil ops (Stencil Fail, Stencil Pass Depth Fail, and Stencil Pass Depth Pass) are KEEP, ZERO, or REPLACE, the stencil buffer is not read.

3DSTATE_WM_DEPTH_STENCIL

28:26	**Stencil Pass Depth Fail Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW] This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth pass fails.
25:23	**Stencil Pass Depth Pass Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW] This field specifies the operation to perform on the Stencil Buffer when the (front face) stencil test passes but the depth test passes.
22:20	**Backface Stencil Test Function** Project: All Format: 3D_Compare_Function
19:17	**Backface Stencil Fail Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW]
16:14	**Backface Stencil Pass Depth Fail Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW] This field specifies the operation to perform on the Stencil Buffer when the stencil test passes but the depth pass fails.
13:11	**Backface Stencil Pass Depth Pass Op** Project: All Format: 3D_Stencil_Operation [CHV, BSW] This field specifies the operation to perform on the Stencil Buffer when the stencil test passes and the depth pass passes (or is disabled).
10:8	**Stencil Test Function** Project: All Format: 3D_Compare_Function This field specifies the comparison function used in the (front face) StencilTest function.
7:5	**Depth Test Function** Project: All Format: 3D_Compare_Function Specifies the comparison function used in DepthTest function.

3DSTATE_WM_DEPTH_STENCIL

Programming Notes
If the Depth Test Function is ALWAYS or NEVER, the depth buffer is not read.

4 **Double Sided Stencil Enable**

Project:	All
Format:	Enable

Enable doubled sided stencil operations.

Value	Name	Description	Project
0h	False	Double Sided Stencil Disabled	All
1h	True	Double Sided Stencil Enabled	All

Programming Notes
• Back-facing primitives have a vertex winding order opposite to the currently selected Front Winding state. • Culling of primitives is not affected by the double sided stencil state • Back-facing primitives will be rendered, honoring all current device state, as though it were a front-facing primitive with no implicitly overloaded state.

3 **Stencil Test Enable**

Project:	All
Format:	Enable

Enables StencilTest function of the Pixel Processing pipeline.

Programming Notes
If any of the render targets are YUV format, this field must be disabled.

2 **Stencil Buffer Write Enable**

Project:	All
Format:	Enable

Enables writes to the Stencil Buffer.

Programming Notes
If this field is enabled, Stencil Test Enable must also be enabled.

1 **Depth Test Enable**

Project:	All
Format:	Enable

Enables the DepthTest function of the Pixel Processing pipeline.

Value	Name	Project
0h	Disable	All
1h	Enable	All

Programming Notes

3DSTATE_WM_DEPTH_STENCIL

		If any of the render targets are YUV format, this field must be disabled.
	0	**Depth Buffer Write Enable** Project: All Format: Enable Enables writes to the Depth Buffer. **Programming Notes** / **Project** A Depth Buffer must be defined before enabling writes to it, or operation is UNDEFINED. This bit must not be set when WM_INT::RT Independent Rasterization Enable is true. — CHV, BSW **Workaround** / **Project** If Depth_Test_Enable = 1 AND Depth_Test_func = EQUAL, the Depth_Write_Enable must be set to 0 — CHV, BSW
2	31:24	**Stencil Test Mask** Project: All Format: U8 This field specifies a bit mask applied to stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.
	23:16	**Stencil Write Mask** Project: All Format: U8 This field specifies a bit mask applied to stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.
	15:8	**Backface Stencil Test Mask** Project: All Format: U8 This field specifies a bit mask applied to backface stencil test values. Both the stencil reference value and value read from the stencil buffer will be logically ANDed with this mask before the stencil comparison test is performed.
	7:0	**Backface Stencil Write Mask** Project: All Format: U8 This field specifies a bit mask applied to backface stencil buffer writes. Only those stencil buffer bits corresponding to bits set in this mask will be modified.

3DSTATE_WM

3DSTATE_WM

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 14h 3DSTATE_WM Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31	**Statistics Enable** Project: All Format: Enable If ENABLED, the Windower and pixel pipeline will engage in statistics gathering. If DISABLED, statistics information associated with this FF stage will be left unchanged. See Statistics Gathering. **Programming Notes** This bit must be disabled if any of these bits is set: 3DSTATE_WM::**Legacy Depth Buffer Clear**, 3DSTATE_WM::**Legacy Hierarchical Depth Buffer Resolve Enable** or 3DSTATE_WM::**Legacy Depth Buffer Resolve Enable**.

3DSTATE_WM

30 **Legacy Depth Buffer Clear Enable**

Project:	All
Format:	Enable

When set, the depth buffer is initialized as a side-effect of rendering pixels.

Programming Notes

If this field is enabled,

1. the **Depth Test Enable** field in DEPTH_STENCIL_STATE must be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.
3. 3DSTATE_DEPTH_BUFFER::Stencil Write Enable must be set if 3DSTATE_STENCIL_BUFFER::Stencil buffer enable is set. Additionally the following must be set to the correct values.
    1. DEPTH_STENCIL_STATE::Stencil Write Mask must be 0xFF
    2. DEPTH_STENCIL_STATE::Stencil Test Mask must be 0xFF
    3. DEPTH_STENCIL_STATE::Back Face Stencil Write Mask must be 0xFF
    4. DEPTH_STENCIL_STATE::Back Face Stencil Test Mask must be 0xFF

Refer to section 0 "Depth Buffer Clear" for additional restrictions when this field is enabled. If this field is enabled,**Pixel Shader Kill Pixel** must be disabled.

29 **Reserved**

Project:	All
Format:	MBZ

28 **Legacy Depth Buffer Resolve Enable**

Project:	All
Format:	Enable

When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Legacy Depth Buffer Clear** and **Legacy Hierarchical Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

Refer to section 11.5.4.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled. **If Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect.

27 **Legacy Hierarchical Depth Buffer Resolve Enable**

Project:	All
Format:	Enable

When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified

3DSTATE_WM

outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Legacy Depth Buffer Clear** and **Legacy Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled. If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect. **Performance Note:** expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.

26 **Legacy Diamond Line Rasterization**

Project:	All
Format:	Enable

This bit, if ENABLED, indicates that the Windower will rasterize zero width lines using the DX9 rasterization rules. If DISABLED, the Windower will rasterize zero width lines using the DX10 rasterization rules (see Strips Fans chapter).

25:23 **Reserved**

Project:	All
Format:	MBZ

22:21 **Early Depth/Stencil Control**

Project:	CHV, BSW
Format:	U2 Enumerated Type

This field specifies the behavior of early depth/stencil test.

Value	Name	Description	Project
0h	NORMAL	Depth/Stencil Test/Write behaves as if it happens post-shader, however the pixel shader is not necessarily executed if the pixel fails depth or stencil test (this is the legacy behavior)	All
1h	PSEXEC	Depth/Stencil Test/Write behaves as if it happens post-shader, and the pixel shader is executed if the pixel fails depth or stencil test (although pre-shader actions such as primitive inclusion, stipple, etc. will still cause the shader not to execute)	All
2h	PREPS	Depth/Stencil Test/Write behaves as if it happens pre-shader. The pixel shader is not executed if the pixel fails depth or stencil test. Depth and stencil writes occur even if the pixel is killed by the shader or post-shader by alpha test, etc. Depth output by the pixel shader is ignored.	All
3h	Reserved		All

3DSTATE_WM

Programming Notes
The Early Depth/Stencil Control field cannot be set to PREPS (value = 2h) if ForceKillpix = ForceON or Forced Thread Dispatch = ForceON

20:19 **Reserved**

18:17 **Position ZW Interpolation Mode**

Project:	All
Format:	U2 Enumerated Type

This field elects "interpolation mode" associated with the Position Z (source depth) and W coordinates passed in the PS payload when the PS requires Position as input. This field does not determine whether these coordinates are actually included in the payload (see Pixel Shader Requires Depth, Pixel Shader Requires W).

Value	Name	Description	Project
0h	INTERP_PIXEL	Evaluate Z & W at the pixel center or UL corner (as specified by Pixel Location of 3DSTATE_MULTISAMPLE)	All
1h	Reserved		All
2h	INTERP_CENTROID		All
3h	INTERP_SAMPLE		All

Programming Notes
WM_INT::RT Independent Rasterization Enable must be disabled in order to select INTERP_SAMPLE.
MSDISPMODE_PERSAMPLE is required in order to select INTERP_SAMPLE.

16:11 **Barycentric Interpolation Mode**

Project:	All
Format:	Enable[6]

Controls which barycentric interpolation terms must be passed into the pixel shader kernel. Bit 0: Perspective Pixel Location barycentric is required Bit 1: Perspective Centroid barycentric is required Bit 2: Perspective Sample barycentric is required Bit 3: Non-perspective Pixel Location barycentric is required Bit 4: Non-perspective Centroid barycentric is required Bit 5: Non-perspective Sample barycentric is required

Programming Notes
If contiguous dispatch modes are enabled, only bit 3 (non-perspective pixel location) can be set, all other bits in this field must be zero. Pixel Location below refers to either the upper left corner or pixel center depending on the **Pixel Location** state of 3DSTATE_MULTISAMPLING). MSDISPMODE_PERSAMPLE is required in order to select Perspective Sample or Non-perspective Sample barycentric coordinates.

10 **Reserved**

Project:	All
Format:	MBZ

3DSTATE_WM

9:8 **Line End Cap Antialiasing Region Width**

Project:	All
Format:	U2

This field specifies the distances over which the coverage of anti-aliased line end caps are computed.

Value	Name	Description	Project
0h	0.5 pixels	0.5 pixels	All
1h	1.0 pixels	1.0 pixels	All
2h	2.0 pixels	2.0 pixels	All
3h	4.0 pixels	4.0 pixels	All

7:6 **Line Antialiasing Region Width**

Project:	All
Format:	U2

This field specifies the distance over which the anti-aliased line coverage is computed.

Value	Name	Description	Project
0h	0.5 pixels	0.5 pixels	All
1h	1.0 pixels	1.0 pixels	All
2h	2.0 pixels	2.0 pixels	All
3h	4.0 pixels	4.0 pixels	All

5 **Reserved**

Project:	All
Format:	MBZ

4 **Polygon Stipple Enable**

Project:	All
Format:	Enable

Enables the Polygon Stipple function.

3 **Line Stipple Enable**

Project:	All
Format:	Enable

Enables the Line Stipple function.

2 **Point Rasterization Rule**

Project:	All
Format:	3D_RasterizationRule

This field specifies the rasterization rules to be applied whenever the edges of a point primitive fall exactly on a pixel sampling point.

Value	Name	Description	Project

3DSTATE_WM					
		0h	RASTRULE_UPPER_LEFT	To match "normal" upper left rules for surface primitives	All
		1h	RASTRULE_UPPER_RIGHT	To match OpenGL point rasterization rules (round to + infinity, where this is the upper right direction wrt OpenGL screen origin of lower left).	All
	1:0	**Reserved**			

3DSTATE_WM_HZ_OP

3DSTATE_WM_HZ_OP

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

This command provides for clearing Z and/or stencil or resolving either HZ buffer or Z buffer.

Programming Notes

3DSTATE_MULTISAMPLE packet must be used prior to this packet to change the Number of Multisamples. This packet must not be used to change Number of Multisamples in a rendering sequence.
3DSTATE_RASTER if used must be programmed prior to using this packet.

This command does support predication from the use of the MI_PREDICATE register. To predicate depth clears and resolves on [CHV, BSW] you must fall back to using the 3D_PRIMITIVE or GPGPU_WALKER commands.

As this command generates an implicit rectangle, SW must make sure any MMIO register writes following WM_HZ_OP must be preceded by **PIPE_CONTROL** with **Command Streamer Stall Enable** bit set.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 0h 3DSTATE_PIPELINED Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 52h 3DSTATE_WM_HZ_OP Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Dword Length** Default Value: 03h Excludes Dword (0,1) Project: All Format: =n

3DSTATE_WM_HZ_OP

		Total Length - 2
1	31	**Stencil Buffer Clear Enable** Project: All Format: Enable When set, the stencil buffer is initialized. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Resolve Enable** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Stencil Write Enable must be set if 3DSTATE_STENCIL_BUFFER::Stencil buffer enable is set.
	30	**Depth Buffer Clear Enable** Project: All Format: Enable When set, the depth buffer is initialized. **Programming Notes** If this field is enabled, 1. the **Depth Buffer Resolve Enable** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled. 2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.
	29	**Scissor Rectangle Enable** Project: All Format: Enable Enables operation of Scissor Rectangle. **Programming Notes** In order get the functionality right if this bit is disabled, driver must clip the clear rectangle to scissor rectangle if scissor test is enabled before clearing. **Workaround** / **Project** Workaround: Due to a Hardware issue this bit must not be set. / CHV, BSW
	28	**Depth Buffer Resolve Enable** Project: All Format: Enable When set, the depth buffer is made to be consistent with the hierarchical depth buffer as a side-effect of rendering pixels. This is

3DSTATE_WM_HZ_OP

intended to be used when the depth buffer is to be used as a surface outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Depth Buffer Clear** and **Hierarchical Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

Refer to section 11.5.4.2 "Depth Buffer Resolve" for additional restrictions when this field is enabled. If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect.

27 **Hierarchical Depth Buffer Resolve Enable**

Project:	All
Format:	Enable

When set, the hierarchical depth buffer is made to be consistent with the depth buffer as a side-effect of rendering pixels. This is intended to be used when the depth buffer has been modified outside of the 3D rendering operation.

Programming Notes

If this field is enabled,

1. the **Depth Buffer Clear** and **Depth Buffer Resolve Enable** fields must both be disabled.
2. 3DSTATE_DEPTH_BUFFER::Depth Write Enable must be set.

Refer to section 11.5.4.3 "Hierarchical Depth Buffer Resolve" for additional restrictions when this field is enabled. If **Hierarchical Depth Buffer Enable** is disabled, enabling this field will have no effect. **Performance Note:** expect the hierarchical depth buffer's impact on performance to be reduced for some period of time after this operation is performed, as the hierarchical depth buffer is initialized to a state that makes it ineffective. Further rendering will tend to bring the hierarchical depth buffer back to a more effective state.

26 **Pixel Position Offset Enable**

Project:	All
Format:	U1 Enumerated Type

Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions.

Programming Notes

Setting this field along with setting the Pixel Location to upper left and number of multisamples to greater than one will cause the device to offset pixel postions by 0.5 both in horizontal and vertical

3DSTATE_WM_HZ_OP

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

25 **Full Surface Depth Clear**

Project:	All
Format:	Enable

Programming Notes

Setting this field to "1" along with "Depth buffer clear" will cause all the pixels/samples in an 8x4 block in the HIZ buffer to be cleared. If "Stc-buffer clear" is also set, then all pixels/samples in a 8x8 block of STC buffer will be cleared to the stc-ref value. Software must set this only when the APP requires the entire Depth surface to be cleared. Setting this field to "1" for STC-buffer only clear without "depth buffer clear" will cause all the pixels/samples in an 8x8 block in the STC buffer to get the stc-ref value.

24 **Reserved**

Project:	All
Format:	MBZ

23:16 **Stencil Clear Value**

Format:	U8.0

This field specifies the stencil clear value.

15:13 **Number of Multisamples**

Project:	All
Format:	U3 Enumerated Type

This field specifies how many samples/pixel exist in the Depth Buffer and Stencil buffers, as log2(#samples).

Value	Name	Description	Project
0h	1	1 sample/pixel	All
1h	2	2 samples/pixel	All
2h	4	4 samples/pixel	All
3h	8	8 samples/pixel	All
4h	Reserved		CHV, BSW
5h-7h	Reserved		All

3DSTATE_WM_HZ_OP

	12:0	**Reserved** Project: All
2 **Programming Notes:** The clear rectangle x and y min and max values must be shifted by the LOD level; i.e. the hardware does not include the LOD in this function. Hence to clear any particular X, Y from the base level, to clear the contents at level "LOD" use (X»LOD) and (Y»LOD).	31:16	**Clear Rectangle Y Min** Project: All Format: U16 in Pixels from Depth Buffer origin (upper left corner) Specifies Ymin value of (inclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with Y coordinates less than Ymin will not be affected. **Value** | **Name** | **Project** [0,16383] | (Device ignores bits 31:30) | CHV, BSW
	15:0	**Clear Rectangle X Min** Project: All Format: U16 in Pixels from Depth Buffer origin (upper left corner) Specifies Xmin value of (inclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with X coordinates less than or equal to Xmin will not be affected. **Value** | **Name** | **Project** [0,16383] | (Device ignores bits 15:14) | CHV, BSW
3 **Programming Notes:** See the programming note in the previous DWORD for the Min values. The clear rectangle x and y min and max values must be shifted by the LOD level; i.e. the hardware does not include the LOD in this function. Hence to clear any particular X, Y from the base level, to clear the contents at level "LOD" use (X»LOD) and (Y»LOD). Hence the max values must be less than or equal to: (Surface Width » LOD) and (Surface Height » LOD) for X Max and Y Max respectively.	31:16	**Clear Rectangle Y Max** Project: All Format: U16 in Pixels from Depth Buffer origin (lower right corner) Specifies Ymax value of (exclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with Y coordinates greater than Ymax will be not be cleared. **Value** | **Name** | **Project** [0,16383] | (Device ignores bits 31:30) | CHV, BSW
	15:0	**Clear Rectangle X Max** Project: All Format: U16 in Pixels from Depth Buffer origin (lower right corner) Specifies Xmax value of (exclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with X coordinates greater than or equal to Xmax will be not be affected. **Value** | **Name** | **Project** [0,16383] | (Device ignores bits 15:14) | CHV, BSW
4	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Sample Mask**

3DSTATE_WM_HZ_OP

Project:	All

Format: Right-justified bitmask (Bit 0 = Sample0). Number of bits that are used is determined by Num Multisamples (3DSTATE_WM_HZ_OP)

A per-multisample-position mask state variable that is immediately and unconditionally ANDed with the sample coverage mask as part of the rasterization process. This mask is applied prior to centroid selection.

Programming Notes

If Number of Multisamples is NUMSAMPLES_1, bits 15:1 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_2, bits 15:2 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_4, bits 15:4 of this field will be zeroed by HW.If Number of Multisamples is NUMSAMPLES_8, bits 15:8 of this field will be zeroed by HW.

A64 Byte Scattered Write MSD

MSD1W_A64_BS - A64 Byte Scattered Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Scattered R/W
Group:	Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Project: All Format: Opcode A64 Scattered Write message
	13	**Reserved**
	12	**Reserved**
	11:10	**Data Elements** Project: All Format: MDC_A64_DS [CHV, BSW] Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 0h Project: All Format: Opcode Byte Read/Write subtype
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Scattered Read MSD

MSD1R_A64_DWS - A64 Dword Scattered Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Scattered R/W
Group:	DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 10h Project: All Format: Opcode A64 Scattered Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**Reserved**
	11:10	**Data Elements** Project: All Format: MDC_A64_DS [CHV, BSW] Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Project: All Format: Opcode Dword Read/Write subtype
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Scattered Write MSD

MSD1W_A64_DWS - A64 Dword Scattered Write MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Project: All Format: Opcode A64 Scattered Write message
	13	**Reserved** Project: All Format: MBZ Ignored
	12	**Reserved** Project: CHV, BSW Format: Ignore Ignored
	11:10	**Data Elements** Project: All Format: MDC_A64_DS [CHV, BSW] Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 1h Project: All Format: Opcode Dword Read/Write subtype

MSD1W_A64_DWS - A64 Dword Scattered Write MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI2_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

		Workaround CMPWR_2W Operation is not supported in A64 SIMD4x2.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI3_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

		Workaround CMPWR_2W is not supported by A64 SIMD4x2.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI1_4x2 - A64 Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI2 - A64 Dword Untyped Atomic Integer Binary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI3 - A64 Dword Untyped Atomic Integer Trinary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 0h Project: All Format: Opcode Operations are on 32-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_DWAI1 - A64 Dword Untyped Atomic Integer Unary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Hword Block Read MSD

MSD1R_A64_HWB - A64 Hword Block Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Project: All Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 3h Project: All Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_HW [CHV, BSW] Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Hword Block Write MSD

MSD1W_A64_HWB - A64 Hword Block Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Project: All Format: Opcode A64 Hword Block Write message
	13	**Reserved** Project: All Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 3h Project: All Format: Opcode Hword Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_HW [CHV, BSW] Specifies the number of contiguous Hwords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Oword Block Read MSD

MSD1R_A64_OWB - A64 Oword Block Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Project: All Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 0h Project: All Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Oword Block Write MSD

MSD1W_A64_OWB - A64 Oword Block Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Project: All Format: Opcode A64 Oword Block Write message
	13	**Reserved** Project: All Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 0h Project: All Format: Opcode Oword Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Oword Dual Block Read MSD

MSD1R_A64_OWDB - A64 Oword Dual Block Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Project: All Format: Opcode A64 Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**A64 Block Message Subtype** Default Value: 2h Project: All Format: Opcode Oword Dual Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Oword Dual Block Write MSD

MSD1W_A64_OWDB - A64 Oword Dual Block Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Block R/W
Group:	OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 15h Project: All Format: Opcode A64 Oword Block Write message
	13	**Reserved** Project: All Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 2h Project: All Format: Opcode Oword Dual Block Read/Write subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_OWD Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Oword Unaligned Block Read MSD

MSD1R_A64_OWUB - A64 Oword Unaligned Block Read MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Block R/W
Group: OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 14h Project: All Format: Opcode A64 Oword Block Read message
	13	**Reserved** Project: All Format: MBZ Ignored
	12:11	**A64 Block Message Subtype** Default Value: 1h Project: All Format: Opcode Oword Unaligned Block Read subtype
	10:8	**Data Elements** Project: All Format: MDC_A64_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Scattered Write MSD

MSD1W_A64_QWS - A64 Qword Scattered Write MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Scattered R/W
Group: QW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 1Ah Project: All Format: Opcode A64 Scattered Write message
	13	**Reserved** Project: All Format: MBZ Ignored
	12	**Reserved** Project: CHV, BSW Format: Ignore Ignored
	11:10	**Data Elements** Project: All Format: MDC_A64_DS [CHV, BSW] Specifies the number of data elements to be read or written
	9:8	**A64 Scattered Message Subtype** Default Value: 2h Project: All Format: Opcode Qword Read/Write subtype

MSD1W_A64_QWS - A64 Qword Scattered Write MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI2_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Specifies the atomic integer operation to be performed. **Workaround** CMPWR_2W is not supported in A64 SIMD4x2

MSD1R_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI3_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

		Workaround CMPWR_2W is not supported in A64 SIMD4x2.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 13h Project: All Format: Opcode A64 Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI1_4x2 - A64 Qword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI2 - A64 Qword Untyped Atomic Integer Binary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary with Return Data Operation MSD

		Workaround CMPWR_2W is not supported in A64 QWord SIMD8.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Qword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: CHV, BSW Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI3 - A64 Qword Untyped Atomic Integer Trinary Write Only Operation MSD

		Workaround CMPWR_2W is not supported in A64 QWord SIMD8.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1R_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary with Return Data Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Qword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] The message forbids a header
	18:14	**Message Type** Default Value: 12h Project: All Format: Opcode A64 Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Data Width** Default Value: 1h Project: All Format: Opcode Operations are on 64-bit integers
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.

MSD1W_A64_QWAI1 - A64 Qword Untyped Atomic Integer Unary Write Only Operation MSD

	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Untyped Surface Read MSD

MSD1R_A64_US - A64 Untyped Surface Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 11h Project: All Format: Opcode A64 Untyped Surface Read message
	13:12	**SIMD Mode** Project: All Format: MDC_SM3 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Project: All Format: MDC_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

A64 Untyped Surface Write MSD

MSD1W_A64_US - A64 Untyped Surface Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHF [CHV, BSW] Indicates that the message forbids a header
	18:14	**Message Type** Default Value: 19h Project: All Format: Opcode A64 Untyped Surface Write message
	13:12	**SIMD Mode** Project: All Format: MDC_SM3 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Project: All Format: MDC_UW_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_STATELESS [CHV, BSW] Specifies the message is stateless

Addition

add - Addition

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The add instruction performs component-wise addition of src0 and src1 and stores the results in dst. Addition of two floating-point numbers follows rules in add (IEEE mode) or add (ALT mode).

Format: [(pred)] add[.cmod] (exec_size) dst src0 src1

Programming Notes

Use a source modifier with add to implement subtraction.

Syntax

[(pred)] add[.cmod] (exec_size) reg reg reg [(pred)] add[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] + src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*B,*W,*D	F	
F	F	
DF	DF	CHV, BSW
HF	HF	CHV, BSW
*B,*W,*D	HF	CHV, BSW
*W,*D,*Q	*W,*D,*Q	CHV, BSW
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]

add - Addition

	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Addition with Carry

addc - Addition with Carry

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The addc instruction performs component-wise addition of src0 and src1 and stores the results in dst; it also stores the carry into acc. If the operation produces a carry out, 0x00000001 is stored in acc, else 0x00000000 is stored in acc.

Format: [(pred)] addc[.cmod] (exec_size) dst src0 src1

Restriction

AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

[(pred)] addc[.cmod] (exec_size) reg reg reg [(pred)] addc[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] + src1.chan[n]; acc.chan[n] = carry(src0.chan[n] + src1.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Arithmetic Shift Right

asr - Arithmetic Shift Right

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

Perform component-wise arithmetic right shift of the bits in src0 by the shift count indicated in src1, storing the results in dst. If src0 has a signed type, insert copies of src0's sign bit in the number of MSBs indicated by the shift count. Otherwise insert 0 bits. In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type. For positive values, this operation is src0 / 2shiftCount and for negative values, this operation is src0 / 2shiftCount - 1.

Format: [(pred)] asr[.cmod] (exec_size) dst src0 src1

Programming Notes

If src0 is -1, the result is -1 regardless of the shift count.

For unsigned src0 types, asr and shr produce the same result.

Syntax

[(pred)] asr[.cmod] (exec_size) reg reg reg [(pred)] asr[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) { shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F if (src0.chan[n] >= 0) { dst.chan[n] = src0.chan[n] » shiftCnt; } else { int maskLSB = pow(2, shiftCnt) - 1; if ( maskLSB & src0.chan[n] == 0 ) { dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt); } else { dst.chan[n] = sign(src0.chan[n]) * ((abs)src0.chan[n] » shiftCnt) - 1; } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*W,*D,*Q	*W,*D,*Q	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]

asr - Arithmetic Shift Right		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Average

avg - Average

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The avg instruction performs component-wise integer average of src0 and src1 and stores the results in dst. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic right shift to this intermediate value.

Format: The avg instruction performs component-wise integer average of src0 and src1 and stores the results in dst. An integer average uses integer upward rounding. It is equivalent to increment one to the addition of src0 and src1 and then apply an arithmetic right shift to this intermediate value.

Syntax

[(pred)] avg[.cmod] (exec_size) reg reg reg [(pred)] avg[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = (src0.chan[n] + src1.chan[n] + 1) » 1; // Use arithmetic shift right. } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Bit Field Extract

bfe - Bit Field Extract

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Component-wise extract a bit field from src2 using the bit field width from src0 and the bit field offset from src1. Store the extracted bit field value in the low bits of dst and sign extend (if D type) or zero extend (if UD type). The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f. If width is zero, the result is zero. If offset + width > 32 then the extracted bit field is bits offset to 31 of src2, extracting only 32 - offset bits, less than width as the bit field cannot extend past the MSB of the source value. Otherwise extract width bits extending from bit positions offset to offset + width - 1.

Format: [(pred)] bfe (exec_size) dst src0 src1 src2

Restriction

No accumulator access, implicit or explicit.

All three-source instructions have certain restrictions, described in Instruction Formats [CHV, BSW].

Syntax

[(pred)] bfe (exec_size) reg reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD width = src0.chan[n][4:0]; UD offset = src1.chan[n][4:0]; if ( width == 0 ) { dst.chan[n] = 0x00000000; } else if ( (width + offset) < 32 ) { dst.chan[n] = src2.chan[n] « (32 - width - offset); if (src2 is signed) { dst.chan[n] = dst.chan[n] » (32 - width); // pad sign bit of dst.chan } else { dst.chan[n] = dst.chan[n] » (32 - width); // pad 0 } } else { if ( src2 is signed ) { dst.chan[n] = src2.chan[n] » offset; // pad sign bit } else { dst.chan[n] = src2.chan[n] » offset; // pad 0 } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105	**Reserved** Format: MBZ

bfe - Bit Field Extract

	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved** Project: CHV, BSW Format: MBZ
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved** Format: MBZ
	34	**Reserved**

bfe - Bit Field Extract

	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Bit Field Insert 1

bfi1 - Bit Field Insert 1

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The bfi1 instruction is the first instruction in a two-instruction macro for bfi (Bit Field Insert). The bfi1 instruction component-wise generates mask with control from src0 and src1 and stores the results in dst. The mask is used in the bfi2 instruction to generate the final result of bfi. Create a bit mask corresponding to the bit field width and offset in src0 and src1. Store the bit mask in dst. The mask has all bits in the bit field set to 1 and all other bits as 0. The width and offset values are from the low five bits of src0 and src1 respectively, or src0 & 0x1f and src1 & 0x1f. If width is zero, the result is zero. The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value. bfi dst src0 src1 src2 src3 // Translates to these two instructions: bfi1 dst src0 src1 bfi2 dst dst src2 src3

Format: [(pred)] bfi1 (exec_size) dst src0 src1

Programming Notes

No accumulator access, implicit or explicit.

Syntax

[(pred)] bfi1 (exec_size) reg reg reg [(pred)] bfi1 (exec_size) reg reg imm32

Pseudocode

Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { UD width = src0.chan[n][4:0]; UD offset = src1.chan[n][4:0]; dst = ((1 « width) - 1) « offset; } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]

bfi1 - Bit Field Insert 1		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Bit Field Insert 2

bfi2 - Bit Field Insert 2

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The bfi2 instruction is the second instruction in a two-instruction macro for bfi (Bit Field Insert). The bfi2 instruction component-wise performs the bitfield insert operation on src1 and src2 based on the mask in src0. Use the mask in src0 to take a bit field value from the low bits of src1 and combine it with the value from src2 (so src2 provides all bits other than those masked out and replaced by the bit field value). Store the result in dst. The bfi macro has four source operands: src0 - bit field width in low five bits, src1 - bit field offset/starting bit position in low five bits, src2 - bit field value to insert, using only the number of least significant bits given by width in src0, and src3 - overall value into which the bit field is inserted, providing all bits other than the inserted bits for the result value. bfi dst src0 src1 src2 src3 // Translates to these two instructions: bfi1 dst src0 src1 bfi2 dst dst src2 src3

Format: [(pred)] bfi2 (exec_size) dst src0 src1 src2

Restriction	Project
No accumulator access, implicit or explicit.	
All three-source instructions have certain restrictions, described in Instruction Formats [CHV, BSW].	CHV, BSW

Syntax

[(pred)] bfi2 (exec_size) reg reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD offset = LZD(reverse(src0.chan[n]))-1; // offset is the number of LSB zero bits below the bit mask which has all 1s. // width (implied by the logic) is the number of 1 bits in the mask value, which should be all 1s. dst.chan[n] = ((src1.chan[n] « offset) & src0.chan[n]) | (src2.chan[n] & ! src0.chan[n]); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD
D	D

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]

bfi2 - Bit Field Insert 2

	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved** Format: MBZ
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved** Format: MBZ

bfi2 - Bit Field Insert 2

	34	**Reserved** Format: MBZ
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Bit Field Reverse

bfrev - Bit Field Reverse

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The bfrev instruction component-wise reverses all the bits in src0 and stores the results in dst.

Format: [(pred)] bfrev (exec_size) dst src0

Restriction

No accumulator access, implicit or explicit.

Syntax

[(pred)] bfrev (exec_size) reg reg [(pred)] bfrev (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { for ( idx = 0; idx < 32; idx++ ) {
dst.chan[n][idx] = src0.chan[n][31-idx]; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Branch Converging

brc - Branch Converging

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

Description

The brc instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if all channels are branched away. UIP should reference the instruction where all channels are expected to come together. JIP should reference the end of the innermost conditional block.

In GEN binary, JIP and UIP use locations src1 and src0 respectively when immediate and location src0 when reg64, where reg64 is accessed as paired DWord (regioning being <2;2,1>). dst must be IP. When the offsets are immediate, src0 regfile must be immediate.

Format: [(pred)] brc (exec_size) JIP UIP

Restriction

A brc instruction cannot use the Switch instruction option.

Syntax

[(pred)] brc (exec_size) imm32 imm32 [(pred)] brc (exec_size) reg64

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn[n] ) { PcIP[n] = IP + UIP; } else { PcIP[n] = IP + 1; } } if ( all PcIP != IP + 1 ) { // for all channels Jump(IP + JIP); }
```

Errata	Description
	A brc instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier	Src Types
Y	N	N	N	D

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.

brc - Branch Converging		
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Branch Diverging

brd - Branch Diverging

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description
The brd instruction redirects the execution forward or backward to the instruction pointed by (current IP + offset). The jump will occur if any channels are branched away.
In GEN binary, JIP is at location src1 when immediate and at location src0 when reg32, where reg32 is accessed as a scalar DWord. The ip register must be used (for example, by the assembler) as dst.
Format: [(pred)] brd (exec_size) JIP

Restriction	Project
A brd instruction cannot use the Switch instruction option.	CHV, BSW

Syntax	Project
[(pred)] brd (exec_size) imm32 [(pred)] brd (exec_size) reg32	CHV, BSW

Pseudocode
Evaluate(WrEn); for (n = 0; n < 32; n++) { if (WrEn[n]) { PcIP[n] = IP + JIP; } else { PcIP[n] = IP + 1; } } if (any PcIP == ExIP + JIP) { // any channel Jump(ExIP + JIP); }

Errata	Description	Project
	A brd instruction must not be followed by any instruction requiring register indirect access on source operands.	CHV, BSW

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types
D

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW

brd - Branch Diverging

	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Break

break - Break

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description
The break instruction is used to early-out from the inner most loop, or early out from the inner most switch block. When used in a loop, upon execution, the break instruction terminates the loop for all execution channels enabled. If all the enabled channels hit the break instruction, jump to the instruction referenced by JIP. JIP should be the offset to the end of the inner most conditional or loop block, UIP should be the offset to the while instruction of the loop block. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case
The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.
Format: [(pred)] break (exec_size) JIP UIP

Restriction
The execution size of break instruction must be same as execution size of the corresponding while instruction.
The execution size of break instruction must be same as execution size of the corresponding while instruction.

Syntax	Project
[(pred)] break (exec_size) imm16 imm16	CHV, BSW

Pseudocode
Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.channel[n]) { PcIP[n] = IP + UIP; else { PcIP[n] = IP + 1; } } if (PcIP != (IP + 1)) { // all channels Jump(IP + JIP); }

Errata	Description	Project
	A break instruction must not be followed by any instruction requiring register indirect access on source operands.	CHV, BSW

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.

break - Break		
	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Byte Scattered Read MSD

MSD0R_BS - Byte Scattered Read MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 04h Project: All Format: Opcode Byte Scattered Read message
	13	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	12	**Reserved** Project: All Format: MBZ Ignored
	11:10	**Data Elements** Project: All Format: MDC_DS [CHV, BSW] Specifies the number of Bytes to be read or written per Dword

MSD0R_BS - Byte Scattered Read MSD

	9	**Reserved** Project: All Format: MBZ Ignored
	8	**SIMD Mode** Project: All Format: MDC_SM2 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Byte Scattered Write MSD

MSD0W_BS - Byte Scattered Write MSD

Project:	CHV, BSW
Source:	DataPort 0
Length Bias:	1
Family:	Scattered R/W
Group:	Byte Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Byte Scattered Write message
	13:12	**Reserved** Project: All Format: MBZ Ignored
	11:10	**Data Elements** Project: All Format: MDC_DS [CHV, BSW] Specifies the number of Bytes to be read or written per Dword
	9	**Reserved** Project: All Format: MBZ Ignored

MSD0W_BS - Byte Scattered Write MSD

	8	**SIMD Mode** Project: All Format: MDC_SM2 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Call

call - Call

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

Description

The call instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the call instruction. If none of the channels jump into the subroutine, the call instruction is treated as a nop. In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register. When SPF is on, the predication control must be scalar.

The following table describes JIP, the jump offset. JIP can be an immediate or register value. When a jump occurs, this value is added to IP pre-increment. For CHV, BSW, in GEN binary, JIP is at location src1 when immediate and at location src0 when in a register. The IP register must be put (for example, by the assembler) at dst location. When offsets are immediate, src0 must be null.

Format: [(pred)] call (exec_size) dst JIP

Restriction

The call instruction must have DWord source and destination type, and the destination must be QWord aligned.

A call instruction must use a Switch

A call instruction must be followed by an instruction that supports Switch. When call takes a jump, the first instruction must have a Switch.

Syntax

[(pred)] call (exec_size) reg imm32 [(pred)] call (exec_size) reg reg32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if (WrEn.chan[n] ) { PcIP[n] = IP + JIP; CallMask[n] = 1; } else { PcIP[n] = IP + 1; CallMask[n] = 0; } } if ( PcIP[n] != (IP + 1) ) { // any channel jumped dst.chan[0] = IP + 1; dst.chan[1] = CallMask; Jump(IP + JIP); }
```

Errata	Description
	A call instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Dst Types
D, UD

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW
	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Call Absolute

calla - Call Absolute

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The calla instruction jumps to a subroutine. It can be predicated or non-predicated. If non-predicated, all enabled channels jump to the subroutine. If predicated, only the channels enabled by PMask jump to the subroutine; the rest of the channels move to the next instruction after the calla instruction. If none of the channels jump into the subroutine, the calla instruction is treated as a nop. In case of a jump, the call instruction stores the return IP onto the first DWord of the destination register and stores the CallMask in the second DWord of the destination register. If SPF is ON, none of the PcIP are updated. When SPF is on, the predication control must be scalar. The difference between calla and call is that calla uses JIP as the IP value rather than adding it to the IP value.

Format: [(pred)] calla (exec_size) dst JIP

Restriction

The calla instruction must have DWord source and destination type, and the destination must be QWord-aligned.

The src0 regioning control must be <2;2,1>

Syntax	Project
[(pred)] calla (exec_size) reg imm32	CHV, BSW

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) { PcIP[n] = JIP; CallMask[n] = 1; } else { PcIP[n] = IP + 1; CallMask[n] = 0; } } if ( PcIP[n] != (IP + 1) ) { // any channel jumped dst.chan[0] = IP + 1; dst.chan[1] = CallMask; Jump(JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Dst Types
D, UD

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW

calla - Call Absolute

	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Compare

cmp - Compare

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The cmp instruction performs component-wise comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional code (excluding NS signal) based on the conditional modifier, and storing the conditional bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results. A conditional modifier must be specified; the conditional modifier field cannot be 0000b. The comparison does not use the NS (NaN source) signals, as described in the Creating Conditional Flags section. Accordingly the conditional modifier should not be .u (unordered). For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst. When any source type is floating-point, the cmp instruction obeys the rules described in the tables in the Floating Point Modes section of the Data Types chapter.

Format: [(pred)] cmp[.cmod] (exec_size) dst src0 src1

Restriction

Accumulator cannot be destination, implicit or explicit. The destination must be a general register or the null register.

Syntax

[(pred)] cmp[.cmod] (exec_size) reg reg reg [(pred)] cmp[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if ( WrEn.chan[n] ) { results[n] = src0.chan[n] - src1.chan[n]; bitMask[n] = Condition(results[n]); dst.chan[n] = bitMask[n]; // All bits for dst channel } } flag# = bitMask;
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*B,*W,*D	F	
F	F	
DF	DF	CHV, BSW
HF	HF	CHV, BSW

cmp - Compare

*B,*W,*D	HF	CHV, BSW
*W,*D,*Q	*W,*D,*Q	CHV, BSW
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Compare NaN

cmpn - Compare NaN

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The cmpn instruction performs component-wise special-NaN comparison of src0 and src1 and stores the results in the selected flag register and in dst. It takes component-wise subtraction of src0 and src1, evaluating the conditional signals including NS based on the conditional modifier, and storing the conditional flag bits in bit-packed form in the destination flag register and all bits of dst channels. If the dst is not null, for the enabled channels, then all bits of the destination channel will contain the flag value for the channel. When the instruction operates on packed word format, one general register may store up to 16 such comparison results. In DWord format, one general register may store up to 8 results. A conditional modifier must be specified; the conditional modifier field cannot be 0000b. More information about the conditional signals used is in the Creating Conditional Flags section. For each enabled channel 0b or 1b is assigned to the appropriate flag bit and 0/all zeros or all ones (e.g, byte 0xFF, word 0xFFFF, DWord 0xFFFFFFFF) is assigned to dst. Min/Max instructions use cmpn to select the destination from the input sources (see the Min Max of Floating Point Numbers section for details).

Format: [(pred)] cmpn[.cmod] (exec_size) dst src0 src1

Restriction

Accumulator cannot be destination, implicit or explicit. The destination must be a general register or the null register.

.l and .ge are the only two conditional modifiers are supported for this instruction.

Syntax

[(pred)] cmpn[.cmod] (exec_size) reg reg reg [(pred)] cmpn[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if ( WrEn.chan[n] ) { results[n] = src0.chan[n] - src1.chan[n]; bitMask[n] = ConditionNaN(results[n]); dst.chan[n][0] = bitMask[n]; // All bits for dst channel } } flag# = bitMask;
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*B,*W,*D	F	
F	F	
DF	DF	CHV, BSW
HF	HF	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Conditional Select

csel - Conditional Select

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The csel instruction selectively moves components in src0 or src1 to the dst based on the result of the compare of src2 with zero. If the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst. The csel instruction provides the function of a cmp followed by sel. The instruction must not be used if cmpn is required. The instruction does not update the flag register.
The comparison follows the same rule as cmp instruction for that data type.

Format: csel (exec_size) dst src0 src1 src2

Syntax

csel[.cmod] (exec_size) reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { bitMask[n] = 0; if ( EMask.chan[n] ) { result[n] = src2.chan[n] - 0; bitMask[n] = Condition(result[n]); if (bitMask[n] = 1) { dst.chan[n] = src0.chan[n]; } else { dst.chan[n] = src1.chan[n]; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]

csel - Conditional Select

	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved** Format: MBZ
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Conditional Send Message

sendc - Conditional Send Message

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The sendc instruction has the same behavior as the send instruction except the following. sendc first checks the dependent threads inside the Thread Dependency Register. There are up to 8 dependent threads in the TDR register. The sendc instruction executes only when all the dependent threads in the TDR register are retired. Wait for dependencies in the TDR Register to clear, then send a message stored in registers starting at src to a shared function identified by exdesc along with control from desc with a general register writeback location at dst.

Format: [(pred)] sendc (exec_size) dst src0 exdesc desc

Restriction

The sendc instruction has the same restrictions as the send instruction.

Pseudocode

```
if ( TDR[7] ... || TDR[2] || TDR[1] || TDR[0] ) { wait; } Evaluate(WrEn); MsgChEnable = WrEn; SourceReg = src0.RegNum; MessageEnqueue(MsgChEnable, ResponseReg, SourceReg, desc, exdesc);
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG [CHV, BSW]
	95:89	**Flags** Format: EU_INSTRUCTION_FLAGS
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B [CHV, BSW]

sendc - Conditional Send Message			
	27:24	**Shared Function ID (SFID)**	
		Format:	SFID [CHV, BSW]
	23:8	**Controls A**	
		Format:	EU_INSTRUCTION_CONTROLS_A [CHV, BSW]
	7	**Reserved**	
		Format:	MBZ
	6:0	**Opcode**	
		Format:	EU_OPCODE [CHV, BSW]

Constant Cache Dword Scattered Read MSD

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD

Project:	CHV, BSW
Source:	Read-Only DataPort
Length Bias:	1
Family:	Scattered R/W
Group:	DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 03h Project: All Format: Opcode Dword Scattered Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**Reserved** Project: All Format: MBZ Ignored
	11:10	**Reserved** Project: CHV, BSW Format: MBZ

MSD_CC_DWS - Constant Cache Dword Scattered Read MSD

	9	**Legacy SIMD Mode** Default Value: 1h Project: All Format: Opcode Must be set for compatibility.
	8	**SIMD Mode** Project: All Format: MDC_SM2 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Constant Cache Oword Block Read MSD

MSD_CC_OWB - Constant Cache Oword Block Read MSD

Project:	CHV, BSW
Source:	Read-Only DataPort
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved** Project: All Format: MBZ Ignored
	17:14	**Message Type** Default Value: 00h Project: All Format: Opcode Oword Block Read Constant Cache message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**Reserved** Project: All Format: MBZ Ignored
	10:8	**Data Elements** Project: All Format: MDC_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read or written

MSD_CC_OWB - Constant Cache Oword Block Read MSD

	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Constant Cache Oword Dual Block Read MSD

MSD_CC_OWDB - Constant Cache Oword Dual Block Read MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1
Family: Block R/W
Group: OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Reserved** Project: All Format: MBZ Ignored
	17:14	**Message Type** Default Value: 02h Project: All Format: Opcode Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:10	**Reserved** Project: All Format: MBZ Ignored
	9:8	**Data Elements** Project: All Format: MDC_DB_OWD [CHV, BSW] Specifies the number of contiguous Owords to be read or written

MSD_CC_OWDB - Constant Cache Oword Dual Block Read MSD

	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Constant Cache Oword Unaligned Block Read MSD

MSD_CC_OWUB - Constant Cache Oword Unaligned Block Read MSD

Project:	CHV, BSW
Source:	Read-Only DataPort
Length Bias:	1
Family:	Block R/W
Group:	OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved** Project: All Format: MBZ Ignored
	17:14	**Message Type** Default Value: 01h Project: All Format: Opcode Oword Unaligned Block Read Constant Cache message
	13:11	**Reserved** Project: All Format: MBZ Ignored
	10:8	**Data Elements** Project: All Format: MDC_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Continue

cont - Continue

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description

The cont instruction disables execution for the subset of channels for the remainder of the current loop iteration. Channels remain disabled until right before the while instuction or right before the condition check code block for the while instruction. If all enabled channels hit this instruction, jump to the instruction referenced by JIP where execution continues. UIP should always reference the loop's associated while instruction. JIP should point to the last instruction of the inner most conditional block if the cont instruction is inside a conditional block. In case of the break instruction directly under the loop, the JIP and the UIP are the same. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.

The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.

Format: [(pred)] cont (exec_size) JIP UIP

Restriction

The execution size must be the same for the while, break, and cont instructions of the same code block.

Syntax

[(pred)] cont (exec_size) imm32 imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.channel[n] ) { if ( PMask[n] ) { // PMask is for all channels enabled for the cont instruction. PcIP[n] = IP + UIP; } else { PcIP[n] = IP + 1; } } } for ( n = exec_size; n < 32; n++ ) { PcIP[n] = IP + 1; } if ( PcIP != (IP + 1) ) { // all channels true Jump(IP + JIP); }
```

Errata	Description
	A cont instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.

cont - Continue

	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Count Bits Set

cbit - Count Bits Set

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The cbit instruction counts component-wise the total bits set in src0 and stores the resulting counts in dst.

Format: [(pred)] cbit (exec_size) dst src0

Restriction

No accumulator access, implicit or explicit.

Syntax

[(pred)] cbit (exec_size) reg reg [(pred)] cbit (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt = 0; UD val = src0.chan[n]; while ( val ) { if ( val & 1 ) { cnt ++; } val = val » 1; } dst.chan[n] = cnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UB, UW, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Dot Product 2

dp2 - Dot Product 2

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The dp2 instruction performs a two-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every third and fourth element of src0 (post-source-swizzle if present) are not involved in the computation. The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements. The dp4 instruction includes all four elements of each vector in the dot product. The dp3 instruction includes the first three elements of each vector in the dot product.

Format: [(pred)] dp2[.cmod] (exec_size) dst src0 src1

Restriction

Execution size cannot be less than 4.

Horizontal strides must be 1.

Source operands cannot be accumulators.

Syntax

[(pred)] dp2[.cmod] (exec_size) reg reg reg [(pred)] dp2[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier	Src Types	Dst Types
Y	Y	Y	Y	F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Dot Product 3

dp3 - Dot Product 3

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The dp3 instruction performs a three-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every fourth element of src0 (post-source-swizzle if present) is not involved in the computation. The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements. The dp4 instruction includes all four elements of each vector in the dot product. The dp2 instruction includes the first two elements of each vector in the dot product.

Format: [(pred)] dp3[.cmod] (exec_size) dst src0 src1

Restriction

Execution size cannot be less than 4.

Horizontal strides must be 1.

Source operands cannot be accumulators.

Syntax

[(pred)] dp3[.cmod] (exec_size) reg reg reg [(pred)] dp3[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]

dp3 - Dot Product 3		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Dot Product 4

dp4 - Dot Product 4

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The dp4 instruction performs a four-wide dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. The dot product of two vectors of equal length is the sum of the products of each pair of corresponding elements.

Format: [(pred)] dp4[.cmod] (exec_size) dst src0 src1

Restriction

Execution size cannot be less than 4.

Horizontal strides must be 1.

Source operands cannot be accumulators.

Syntax

[(pred)] dp4[.cmod] (exec_size) reg reg reg [(pred)] dp4[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2] + src0.chan[n+3] * src1.chan[n+3]; if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Dot Product Homogeneous

dph - Dot Product Homogeneous

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The dph instruction performs a four-wide homogeneous dot product on four-tuple vector basis and storing the same scalar result per four tuple to all four channels in dst. This instruction is similar to dp4 except that every fourth element of src0 (post-source-swizzle if present) is forced to 1.0f. Use the dp4 instruction to do a four-wide dot product that includes all elements of src0 and src1.

Format: [(pred)] dph[.cmod] (exec_size) dst src0 src1

Restriction

Execution size cannot be less than 4.

Horizontal strides must be 1.

Source operands cannot be accumulators.

Syntax

[(pred)] dph[.cmod] (exec_size) reg reg reg [(pred)] dph[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 4 ) { fTmp = src0.chan[n] * src1.chan[n] + src0.chan[n+1] * src1.chan[n+1] + src0.chan[n+2] * src1.chan[n+2] + src1.chan[n+3]; // Use 1.0f in place of src0.chan[n+3]. if ( WrEn.chan[n] ) dst.chan[n] = fTmp; if ( WrEn.chan[n+1] ) dst.chan[n+1] = fTmp; if ( WrEn.chan[n+2] ) dst.chan[n+2] = fTmp; if ( WrEn.chan[n+3] ) dst.chan[n+3] = fTmp; }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Dword Atomic Counter Binary with Return Data Operation MSD

MSD1R_DWAC2 - Dword Atomic Counter Binary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Project: All Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2RS [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Atomic Counter Binary Write Only Operation MSD

MSD1W_DWAC2 - Dword Atomic Counter Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Project: All Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2RS [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Atomic Counter Unary with Return Data Operation MSD

MSD1R_DWAC1 - Dword Atomic Counter Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Project: All Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2RS [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Atomic Counter Unary Write Only Operation MSD

MSD1W_DWAC1 - Dword Atomic Counter Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Bh Project: All Format: Opcode Atomic Counter Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2RS [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Scattered Read MSD

MSD0R_DWS - Dword Scattered Read MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 03h Project: All Format: Opcode Dword Scattered Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12	**Reserved** Project: All Format: MBZ Ignored
	11:10	**Reserved** Project: CHV, BSW Format: MBZ Ignored

MSD0R_DWS - Dword Scattered Read MSD

	9	**Legacy SIMD Mode** Default Value: 1h Project: All Format: Opcode Must be set for compatibility.
	8	**SIMD Mode** Project: All Format: MDC_SM2 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Project: CHV, BSW Format: MDC_BTS_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Scattered Write MSD

MSD0W_DWS - Dword Scattered Write MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Scattered R/W
Group: DW Scattered R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Bh Project: All Format: Opcode Dword Scattered Write message
	13:12	**Reserved** Project: All Format: MBZ Ignored
	11:10	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	9	**Legacy SIMD Mode** Default Value: 1h Project: All Format: Opcode Must be set for compatibility.

MSD0W_DWS - Dword Scattered Write MSD

	8	**SIMD Mode** Project: All Format: MDC_SM2 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	7:0	**Binding Table Index** Project: CHV, BSW Format: MDC_BTS_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD

MSD1R_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD

MSD1W_DWAC2_4x2 - Dword SIMD4x2 Atomic Counter Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD

MSD1R_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD

MSD1W_DWAC1_4x2 - Dword SIMD4x2 Atomic Counter Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Atomic Counter Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header
	18:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Atomic Counter Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD

MSD1W_DWTAI2_4x2 - Dword SIMD4x2 Typed Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWTAI3_4x2 - Dword SIMD4x2 Typed Atomic Integer Trinary Write Only Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Typed Atomic Operation
Group: Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Typed Atomic Operation
Group: Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD

MSD1W_DWTAI1_4x2 - Dword SIMD4x2 Typed Atomic Integer Unary Write Only Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Typed Atomic Operation
Group: Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 07h Project: All Format: Opcode Typed Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_DWAI2_4x2 - Dword SIMD4x2 Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWAI3_4x2 - Dword SIMD4x2 Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_DWAI1_4x2 - Dword SIMD4x2 Untyped Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 03h Project: All Format: Opcode Untyped Atomic Integer Operation SIMD4x2 message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Reserved**
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWTAI2 - Dword Typed Atomic Integer Binary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Binary Write Only Operation MSD

MSD1W_DWTAI2 - Dword Typed Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWTAI3 - Dword Typed Atomic Integer Trinary with Return Data Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWTAI3 - Dword Typed Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3S [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWTAI1 - Dword Typed Atomic Integer Unary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Typed Atomic Operation
Group: Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Typed Atomic Integer Unary Write Only Operation MSD

MSD1W_DWTAI1 - Dword Typed Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Atomic Operation
Group:	Dword Typed Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 06h Project: All Format: Opcode Typed Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**Slot Group** Project: All Format: MDC_SG2 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Binary with Return Data Operation MSD

MSD1R_DWAI2 - Dword Untyped Atomic Integer Binary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Binary Write Only Operation MSD

MSD1W_DWAI2 - Dword Untyped Atomic Integer Binary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Binary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP2 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

MSD1R_DWAI3 - Dword Untyped Atomic Integer Trinary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Trinary Write Only Operation MSD

MSD1W_DWAI3 - Dword Untyped Atomic Integer Trinary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Trinary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP3 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Unary with Return Data Operation MSD

MSD1R_DWAI1 - Dword Untyped Atomic Integer Unary with Return Data Operation MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Untyped Atomic Operation
Group: Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 1h Project: All Format: Opcode Specifies that return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Dword Untyped Atomic Integer Unary Write Only Operation MSD

MSD1W_DWAI1 - Dword Untyped Atomic Integer Unary Write Only Operation MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Atomic Operation
Group:	Dword Untyped Atomic Integer Unary Operation

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 02h Project: All Format: Opcode Untyped Atomic Integer Operation message
	13	**Return Data Control** Default Value: 0h Project: All Format: Opcode Specifies that no return data is sent back to the thread.
	12	**SIMD Mode** Project: All Format: MDC_SM2R [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Atomic Integer Operation** Project: All Format: MDC_AOP1 [CHV, BSW] Specifies the atomic integer operation to be performed.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

End If

endif - End If

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description
The endif instruction terminates an if/else/endif block of code. It restores execution to the channels that were active prior to the if/else/endif block. The endif instruction is also used to hop out of nested conditionals by jumping to the end of the next outer conditional block when all channels are disabled.
The following table describes the 32-bit JIP. In GEN binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand, it is a signed 32-bit number. This value is added to IP pre-increment.
Format: endif JIP

Restriction
Predication is not allowed.
The execution size must be the same for the if, else, and endif instructions of the same code block.

Syntax	Project
endif (exec_size) imm32	CHV, BSW

Pseudocode
Evaluate(WrEn); if (WrEn == 0) { // all channels false Jump(IP + JIP); }

Errata	Description
	An endif instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW

endif - End If

	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Extended Math Function

math - Extended Math Function

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The math instruction performs extended math function on the components in src0, or src0 and src1, and write the output to the channels of dst. The type of extended math function are based on the FC[3:0] encoding in the table below.

Format: [(pred)] math (exec_size) dst src0 src1 <FC>

Restriction
Accumulator access is allowed only for ieee macro functions.
The math instruction does not support indirect addressing modes.
The only supported rounding mode for math instruction is Round to Nearest Even.
INT DIV function does not support SIMD16. INT DIV function does not support simulataneous write to two registers. INT DIV function does not support source modifiers.
The FDIV function is not supported in ALT_MODE.
The math instruction can use GRF or immediates as source. The source formats for immediates must be one of the source formats supported by math operation.
DepCtrl must not be used.
The math instruction must use GRF as destination.
The supported regioning mode for math instruction is align1 and align16. The following restrictions apply for align1 mode: Scalar source is supported. Source and destination horizontal stride must be the same. Regioning must ensure Src.Vstride = Src.Width * Src.Hstride . Source and destination offset must be the same, except the case of scalar source.
Math Operation rules when float and half-floats are mixed between source or between source and destination operands. The half-float operand must be interleaved in the register for align1 and the source and destination register offset must be be the same to DW granularity. For align16, the half-float operand is allowed to be packed.
The execution size must be no more than 8 when half-floats are used in source or destination operand.
The source operand must not span two registers if the destination operand spans just one register Example: Case (a) // Allowed. The source must be strided by 2. the offset is allowed to select between lower/upper 16b math (8) r10:f r11.0<16;8,2>:hf 0x01 math (8) r10:f r11.1<16;8,2>:hf 0x01 math (8) r10:f r11.0<16;8,2>:hf r12.1<16;8,2>:hf 0x09 Case (b) // Allowed. The destination must be strided by 2. The offset is allowed to selecte between lower/upper 16b math (8) r10.0<2>:hf r11.0<8;8,1>:f math (8) r10.1<2>:hf r11.0<8;8,1>:f 0x01 math (8) r10.0<2>:hf r11.0<16;8,2>:hf r12.0<16;8,2>:hf 0x09 Case (c) // Allowed. Destination has stride of 2. The offset is allowed to select between uppoer/lower 16b math (8) r10.0<2>:hf r11.0<8;8,1>:f r12.1<16;8,2>:hf 0x09 math (8) r10.1<2>:hf r11.1<16;8,2>:hf r12.0<8;8,1>:f 0x09 Case (d) // Not Allowed. Destination is half-float but is not interleaved. math (8) r10.0<1>:hf r11.0<8;8,1>:f Case (e) // Not Allowed. Source is half-float but

math - Extended Math Function

not interleaved math (8) r10.0<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf 0x09 Case (f) // Not Allowed. Source operand spans 2 registers while destination spans one register. math (8) r83.8<1>:hf r12.4<4;4,1>:f null 0x02
Math Operation rules when half-floats are used on both source and destination operands. The execution size must be 8. The half-float source must be packed or interleaved. When interleaving, both source and destination must be interleaved. Example: Case (a) // Allowed. The source and destination are packed or interleaved math (8) r10.0:hf r11.0<8;8,1>:hf 0x01 math (8) r10.0<2>:hf r11.0<16;8,2>:hf 0x01 math (8) r10.8:hf r11.0<8;8,1>:hf 0x01 math (8) r10.8<2>:hf r11.0<16;8,2>:hf 0x01
For FDVI and POW, half-float destination data type is not allowed.
For one source math operations src1 must be NULL.

Syntax

[(pred)] math (exec_size) reg reg reg imm4

Pseudocode

```
Evaluate(WrEn);
for (n = 0; n < exec_size; n++) {
    if (WrEn.channel[n] == 1) {
        switch FC[3:0] {
            case 1h:
                dst.channel[n] = rcp(src0.channel[n]);
            case 2h:
                dst.channel[n] = log(src0.channel[n]);
            case 3h:
                dst.channel[n] = exp(src0.channel[n]);
            case 4h:
                dst.channel[n] = sqrt(src0.channel[n]);
            case 5h:
                dst.channel[n] = rsq(src0.channel[n]);
            case 6h:
                dst.channel[n] = sin(src0.channel[n]);
            case 7h:
                dst.channel[n] = cos(src0.channel[n]);
            case 9h: // src0 / src1
                dst.channel[n] = fdiv(src0.channel[n], src1.channel[n]);
            case Ah:
                dst.channel[n] = pow(src0.channel[n], src1/channel[n]);
            case Bh: // src0 / src1
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = quotient;
                dst+1.channel[n] = remainder;
            case Ch:
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = quotient;
            case Dh:
                idiv(src0.channel[n], src1.channel[n]);
                dst.channel[n] = remainder;
            }
        }
    }
}
```

math - Extended Math Function

Predication	Conditional Modifier	Saturation	Source Modifier [CHV, BSW]
Y	N	Y	Y

Src Types	Dst Types	Project
F	F	
D	D	
UD	UD	
F/HF	F/HF	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B [CHV, BSW]
	27:24	**Function Control (FC)** Format: FC [CHV, BSW]
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A [CHV, BSW]
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

Find First Bit from LSB Side

fbl - Find First Bit from LSB Side

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The fbl instruction counts component-wise the number of LSB 0 bits before the first 1 bit in src0, storing that number in dst.

Format: [(pred)] fbl (exec_size) dst src0

Programming Notes

If src0 contains no 1 bits, store 0xFFFFFFFF in dst.

Restriction

No accumulator access, implicit or explicit.

Syntax

[(pred)] fbl (exec_size) reg reg [(pred)] fbl (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt = 0; UD udScalar = src0.chan[n]; while ( (udScalar & 1) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar » 1; } if ( src0.chan[n] == 0x00000000 ) { dst.chan[n] = 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Find First Bit from MSB Side

fbh - Find First Bit from MSB Side

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

If src0 is unsigned, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is signed and positive, the fbh instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is signed and negative, the fbh instruction counts component-wise the leading ones from src0 and stores the resulting counts in dst.

Format: [(pred)] fbh (exec_size) dst src0

Programming Notes

If src0 is zero, store 0xFFFFFFFF in dst.

If src0 is signed and is -1 (0xFFFFFFFF), store 0xFFFFFFFF in dst.

Restriction

No accumulator access, implicit or explicit.

Syntax

[(pred)] fbh (exec_size) reg reg [(pred)] fbh (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD cnt = 0; if ( src0 is unsigned ) { UD udScalar = src0.chan[n]; while ( (udScalar & (1 « 31)) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar « 1; } if ( src0.chan[n] == 0x00000000 ) { dst.chan[n] = 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } else { // src0 is signed. D dScalar = src0.chan[n]; bit cval = dScalar[31]; while ((dScalar & (1 « 31)) == cval && cnt != 32 ) { cnt ++; dScalar = dScalar « 1; } if ( (src0.chan[n] == 0xFFFFFFFF) || (src0.chan[n] == 0x00000000) ) { dst.chan[n] = 0xFFFFFFFF; } else { dst.chan[n] = cnt; } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

Src Types	Dst Types
D, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]

fbh - Find First Bit from MSB Side		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Fraction

frc - Fraction

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The frc instruction computes, component-wise, the truncate-to-minus-infinity fractional values of src0 and stores the results in dst. The results, in the range of [0.0, 1.0], are the fractional portion of the source data. The result is in the range [0.0, 1.0] irrespective of the rounding mode. Floating-point fraction computation follows the rules in the following tables, based on the current floating-point mode.

Format: [(pred)] frc[.cmod] (exec_size) dst src0

Workaround

When the Rounding Mode in cr0.0 is not Round Down, the result from frc must be followed by compare and select instructions to avoid a result of 1.0. Those latter instructions must use the :ud type. For example: cmp.ne.f0.0 null r4:ud 0x3f800000:ud (f0.0)sel r5:f r4:ud 0x3f7fffff:ud

Syntax

[(pred)] frc[.cmod] (exec_size) reg reg [(pred)] frc[.cmod] (exec_size) reg imm32

Pseudocode

Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { dst.chan[n] = src0.chan[n] - floor(src0.chan[n]); } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Goto

goto - Goto

Project:	CHV, BSW
Source:	EuIsa
Length Bias:	4

The goto instruction directs the instruction pointer to the offset specified by the UIP offset or to the next IP based on the BranchCtrl bit in the instruction. The active channels that are predicated on this instruction will take the IP + UIP path when BranchCtrl is set else the channels take IP + 1. The active channels that are not predicated on this instruction will be made inactive and waiting to be joined at the join IP. The join IP is IP + UIP when BranchCtrl is clear else it is the next IP.
When there are no active channels the instruction pointer will move to IP + JIP.
The goto instruction is used in conjunction with a join instruction. A goto deactivates some channels that are reactivated at some program-specified join instruction. See the join instruction for the activation rules.
The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence.
The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer).
If SPF is ON, none of the PcIP are updated.

Format: [(pred)] goto (exec_size) JIP UIP BranchCtrl

Restriction

Cannot have a goto in the body and the corresponding join in the function or the subroutine. Similarly the brd and brc.

Syntax

[(pred)] goto (exec_size) imm32 imm32 BranchCtrl

Pseudocode

Evaluate(WrEn);

```
 for ( n = 0; n < exec size; n++ ) { if ( WrEn.chan[n] ) { // for the predicated
active channels if ( BranchCtrl ) { PcIP[n] = IP + UIP; } else { PcIP[n] = IP +
1; } } else { // join IP, for the active non predicated channels if ( BranchCtrl
) { PcIP[n] = IP + 1; } else { PcIP[n] = IP + UIP; } } } if ( BranchCtrl ) { //
if (PcIP != (IP + UIP) ) { // for all channels if (PcIP != (IP + 1) ) { // for
all channels Jump(IP + JIP); } else { Jump(IP + 1); } } else { Jump(IP + UIP); }
} else { // if (PcIP != (IP + 1) ) { // for all channels Jump(IP + JIP); } else {
Jump(IP + 1); } }
```

Errata	Description
	A goto instruction must not be followed by any instruction requiring register indirect access on source operands.

goto - Goto

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

GPGPU_CSR_BASE_ADDRESS

GPGPU_CSR_BASE_ADDRESS

Project: CHV, BSW
Source: PRM
Length Bias: 2

The GPGPU_CSR_BASE_ADDRESS command sets the base pointers for EU and L3 to Context Save and Restore EU State and SLM for **GPGPU** mid.

Programming Notes	Project
Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance. State and instruction caches are flushed on completion of the flush.	CHV, BSW
SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set prior to programming GPGPU_CSR_BASE_ADDRESS command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies for certain programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: Opcode
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON Format: Opcode
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED Format: Opcode
	23:16	**3D Command Sub Opcode** Default Value: 04h GPGPU_CSR_BASE_ADDRESS Format: Opcode
	15:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length -2 Value: 1h — Name: **[Default]** — Description: Excludes DWord(0,1) — Project: CHV, BSW

GPGPU_CSR_BASE_ADDRESS		
1..2 **Project:** CHV, BSW	63:12	**GPGPU CSR Base Address High** Project: CHV, BSW Format: GraphicsAddress[63:12] Specifies the 256K-byte aligned base address for GPGPU context GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11:0	**Reserved** Project: All Format: MBZ

GPGPU_WALKER

GPGPU_WALKER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Programming Notes

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

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h GPGPU_WALKER Format: OpCode
	23:16	**SubOpcode** Default Value: 05h GPGPU_WALKER SubOp Format: OpCode
	15:11	**Reserved** Format: MBZ
	10	**Indirect Parameter Enable** Format: Enable If set, the values in DW 7, 10, 12 are ignored and replaced by the current values of the corresponding GPGPU_xxx MMIO registers: • GPGPU_DISPATCHDIMX (instead of DW7) • GPGPU_DISPATCHDIMY (instead of DW10) • GPGPU_DISPATCHDIMZ (instead of DW12)
	9	**Reserved** Format: MBZ

GPGPU_WALKER

	8	**Predicate Enable** Format: Enable If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if **PredicateEnable** is set and the Predicate state bit is 0.
	7:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 0Dh / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31:17	**Reserved** Format: MBZ
	16:0	**Indirect Data Length** Format: U17 in bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. If Indirect Data is enabled, it is assumed that CURBE is not being used except for cross-thread constant data. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, the total size of Indirect data must be less than 63,488 (2048 URB lines - 64 lines for Interface Descriptors). **Workaround** / **Project** The indirect payload is limited to 4032 bytes or less. / CHV, BSW
3	31:6	**Indirect Data Start Address** Format: IndirectObjectOffset[31:6] This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address.** Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. It is the 64-byte aligned address of the indirect data. **Value** / **Name** / **Description** [0 - 512MB] / / (Bits 31:29 MBZ)

GPGPU_WALKER

	5:0	**Reserved** Format: MBZ
4	31:30	**SIMD Size** This field determines the size of the payload and the number of bits of the execution mask that are expected. The kernel pointed to by the interface descriptor should match the SIMD declared here. Value / Name / Description: 0 / SIMD8 / 8 LSBs of the execution mask are used 1 / SIMD16 / 16 LSBs used in execution mask 2 / SIMD32 / 32 bits of execution mask used
	29:22	**Reserved** Format: MBZ
	21:16	**Thread Depth Counter Maximum** The maximum value of the thread depth counter. Since the counter starts at 0, the depth is this value + 1. **(Thread_Depth_Max+1)*(Thread_Height_Max+1)*(Thread_Width_Max+1)** must equal **Number of Threads in GPGPU Thread Group** in the Interface Descriptor.
	15:14	**Reserved** Format: MBZ
	13:8	**Thread Height Counter Maximum** Format: U6-1 The maximum value of the thread height counter. The height is this value + 1.
	7:6	**Reserved** Format: MBZ
	5:0	**Thread Width Counter Maximum** Format: U6-1 The maximum value of the thread width counter. The height is this value + 1.
5	31:0	**Thread Group ID Starting X** This is the initial value of the X component of the thread group. When X reaches the maximum value it rolls around to this value.
6	31:0	**Reserved** Format: MBZ
7	31:0	**Thread Group ID X Dimension** The X dimension of the thread group (maximum X is dimension -1)
8	31:0	**Thread Group ID Starting Y** This is the initial value of the Y component of the thread group. When Y reaches the maximum value it rolls around to this value.

GPGPU_WALKER		
9	31:0	**Reserved** Format: MBZ
10	31:0	**Thread Group ID Y Dimension** The Y dimension of the thread group (maximum Y is dimension -1)
11	31:0	**Thread Group ID Starting/Resume Z** This is the initial value of the Z component of the thread group.
12	31:0	**Thread Group ID Z Dimension** The Z dimension of the thread group (maximum Z is dimension -1)
13	31:0	**Right Execution Mask** The execution mask used for threads on the right (maximum X) of the thread group.
14	31:0	**Bottom Execution Mask** The execution mask used for threads on the bottom (maximum Y) of the thread group.

Half Precision HI8DS Render Target Write MSD

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 1h Project: CHV, BSW Format: Opcode Half precision data message
	29	**Reserved** Project: All Format: MBZ Ignored
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.

MSD_RTWH_HI8DS - Half Precision HI8DS Render Target Write MSD

	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
	13	**Reserved**
	12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 3h Project: All Format: Opcode SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Half Precision LO8DS Render Target Write MSD

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Project: All Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Project: CHV, BSW Format: Opcode Half precision data message
	29	**Reserved**
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.

MSD_RTWH_LO8DS - Half Precision LO8DS Render Target Write MSD

	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
	13	**Reserved**
	12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 2h Project: All Format: Opcode SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Half Precision REP16 Render Target Write MSD

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Project: All Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Project: CHV, BSW Format: Opcode Half precision data message
	29	**Reserved** Project: All Format: MBZ Ignored
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
	13	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_REP16 - Half Precision REP16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 1h Project: All Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Half Precision SIMD8 Render Target Write MSD

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

Project: CHV, BSW
Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 1h Project: CHV, BSW Format: Opcode Half precision data message
	29	**Reserved** Project: All Format: MBZ Ignored
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.

MSD_RTWH_SIMD8 - Half Precision SIMD8 Render Target Write MSD

	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
	13	**Reserved**
	12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
	10:8	**Render Target Message Subtype** Default Value: 4h Project: All Format: Opcode SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Half Precision SIMD16 Render Target Write MSD

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Project: All Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 1h Project: CHV, BSW Format: Opcode Half precision data message
	29	**Reserved** Project: All Format: MBZ Ignored
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
	13	**Reserved** Project: CHV, BSW Format: MBZ Ignored
	12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
	11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

MSD_RTWH_SIMD16 - Half Precision SIMD16 Render Target Write MSD

	10:8	**Render Target Message Subtype** Default Value: 0h Project: All Format: Opcode SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Halt

halt - Halt

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description
The halt instruction temporarily suspends execution for all enabled compute channels. Upon execution, the enabled channels are sent to the instruction at (IP + UIP), if all channels are enabled at HALT, jump to the instruction at (IP + JIP). If the halt instruction is not inside any conditional code block, the values of JIP and UIP should be the same. If the halt instruction is inside a conditional code block, the UIP should be the end of the program and the JIP should be the end of the inner most conditional code block. The UIP must point to a HALT Instruction. If SPF is ON, the UIP must be used to update IP; JIP is not used in this case.
The following table describes the two 32-bit instruction pointer offsets. Both the JIP and UIP are signed 32-bit numbers, added to IP pre-increment. In GEN binary, JIP and UIP are at locations src0 and src1 and must be of type DW (signed DWord integer). When the offsets are immediate, src0 regfile must be immediate.
Format: [(pred)] halt (exec_size) JIP UIP

Restriction
dst and src0 must be NULL.

Syntax	Project
[(pred)] halt (exec_size) imm32 imm32	CHV, BSW

Pseudocode
Evaluate(WrEn); for (n = 0; n < 32; n++) { if (WrEn.channel[n]) { PcIP[n] = IP + UIP; else { PcIP[n] = IP + 1; } } if (PcIP != (IP + 1)) { // for all channels Jump(IP + JIP); }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.

halt - Halt		
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

HCP_BSD_OBJECT

HCP_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_BSD_OBJECT command fetches the HEVC bit stream for a slice starting with the first byte in the slice. The bit stream ends with the last non-zero bit of the frame and does not include any zero-padding at the end of the bit stream. There can be multiple slices in a HEVC frame and thus this command can be issued multiple times per frame.

The HCP_BSD_OBJECT command must be the last command issued in the sequence of batch commands before the HCP starts decoding. Prior to issuing this command, it is assumed that all configuration parameters in the HCP have been loaded including workload configuration registers and configuration tables. When this command is issued, the HCP is waiting for bit stream data to be presented to the shift register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 20h HCP_BSD_OBJECT_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: U12 (Excludes Dwords 0, 1). Value: 1h — Name:

HCP_BSD_OBJECT

1 — 31:0 — Indirect BSD Data Length

Project:	CHV, BSW
Format:	U32

Specifies the length in bytes of the bitstream data for the current slice. It includes the first byte of the slice and the last non-zero byte of the in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded.

Value	Name	Description	Project
[0,268435455]	Data_Length_with_28_bits_only	Valid range is only from 0 to 268435455, which is corresponding to lower 28 bits. This restriction is for old project which only use 28 bits data length.	CHV, BSW

2 — 31:29 — Reserved

Format:	MBZ

2 — 28:0 — Indirect Data Start Address

Format:	U29

Specifies the byte-aligned graphics memory starting address of the slice bit stream relative to the **BSD Indirect Object Base Address**.

HCP_IND_OBJ_BASE_ADDR_STATE

HCP_IND_OBJ_BASE_ADDR_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_IND_OBJ_BASE_ADDR_STATE command is used to define the indirect object base address of the stream in graphics memory. This is a frame level command. (Is it frame or picture level?)
This is a picture level state command and is issued in both encoding and decoding processes.

Compressed Header Format

Fields	Bits	
Bin	0	Kernel Binarized Syntax
Probability select	1	0 ->indicates probability 128 1 ->indicates probability 256
	Repeat to pack a Cacheline	

Partition1 and TileSize record

Fields	Bits	
Tile Size	31:0	Partition1 Size is 16-bit value, Tile Size is 32-bit value
AddressOffset	63:32	Cacheline Address Offset to be Modified
Offset	69:64	Byte offset to be Modified
16-bit vs 32-bit update	70	0: Update 16-bit; 1: Update 32-bit
Reserved	511:71	

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h

HCP_IND_OBJ_BASE_ADDR_STATE

	22:16	**Media Instruction Command** Default Value: 3h HCP_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 4h; Name: ; Project: CHV, BSW
1..2 **Project:** All	63:0	**HCP Indirect Bitstream Object Base Address** Project: All Format: SplitBaseAddress4KByteAligned Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the HCP_BSD_OBJECT command for fetching (reading) the compressed Slice Data.
3 **Project:** All	31:0	**HCP Indirect Bitstream Object Memory Address Attributes** Project: All Format: MemoryAddressAttributes [CHV, BSW]
4..5 **Project:** All	63:0	**HCP Indirect Bitstream Object Access Upper Bound** Project: All Format: SplitBaseAddress4KByteAligned Decoder only. This field specifies the 4K-byte aligned maximum memory address access by the indirect data object in the HCP_BSD_OBJECT command for the slice bit stream. Indirect data accessed at this address or greater will cause the HCP to stop issuing requests to the GAC and the BSP VLD will then only receive zeros until a slice done is received. Setting this field to 0 will cause this range to be ignored by the HCP.
6..13 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Format: MBZ

HCP_PIC_STATE

HCP_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a picture level command and is issued only once per workload for both encoding and decoding processes.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 10h HCP_PIC_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 11h Name:
1	31:26	**Reserved** Project: CHV, BSW Format: MBZ

HCP_PIC_STATE

25:16 FrameHeightInMinCbMinus1

Project:	CHV, BSW
Format:	U10

Specifies the height of each decoded picture in units of minimum coding block size.

Value	Name
[0-4122]	

Programming Notes	Project
The decoded picture height in units of luma samples equals • (FrameHeightInMinCbMinus1 + 1) * • (1 « (log2_min_coding_block_size_minus3 + 3))	
In CHV, BSW, the maximum frame height being support is 7968 pixels only	CHV, BSW

14:10 Reserved

9:0 FrameWidthInMinCbMinus1

Project:	CHV, BSW
Format:	U10

Specifies the width of each decoded picture in units of minimum coding block size.

Value	Name
[0-4122]	

Programming Notes
The decoded picture width in units of luma samples equals • (FrameWidthInMinCbMinus1 + 1) * • (1 « (log2_min_coding_block_size_minus3 + 3))

2 31:12 Reserved

Project:	
Format:	MBZ

11:10 MaxPCMSize

Format:	U2

Specifies the largest allowed PCM coding block size.

Value	Name
3	64x64
2	32x32
1	16x16
0	8x8

HCP_PIC_STATE

9:8 **MinPCMSize**

Format:	U2

Specifies the smallest allowed PCM coding block size.

Value	Name
3	64x64
2	32x32
1	16x16
0	8x8

7:6 **MaxTUSize**

Format:	U2

Specifies the largest allowed transform block size.

Value	Name
3	32x32
2	16x16
1	8x8
0	4x4

5:4 **MinTUSize**

Format:	U2

Specifies the smallest allowed transform block size.

Value	Name
3	32x32
2	16x16
1	8x8
0	4x4

3:2 **CtbSize (LCUSize)**

Format:	U2

Specifies the coding tree block size.

Value	Name
3	64x64
2	32x32
1	16x16
0	illegal/reserved

Programming Notes
LCU is restricted based on the picture size.

HCP_PIC_STATE

MinCUSize (DWord 3, bits 1:0)

Format:	U2

Specifies the smallest coding block size.

Value	Name
3	64x64
2	32x32
1	16x16
0	8x8

Reserved (DWord 3, bits 31:3)

Project:	
Format:	MBZ

InsertTestFlag (DWord 3, bit 2)

Format:	U1

Value	Name
0h	**[Default]**
1h	

Programming Notes
Encoder only. CABAC 0 Word Insertion Test Enable (Encoder Only)This bit will modify CABAC K equation so that a positive K value can be generated easily. This is done for validation purpose only. In normal usage this bit should be set to 0.Regular equation for generating 'K' value when CABAC 0 Word Insertion Test Enable is set to 0.K = { [((96 * pic_bin_count()) - (RawMinCUBits * PicSizeInMinCUs *3) + 1023) / 1024] - bytes_in_picture } / 3Modified equation when CABAC 0 Word Insertion Test Enable bit set to 1.K = { [((1536 * pic_bin_count()) - (RawMinCUBits * PicSizeInMinCUs *3) + 1023) / 1024] - bytes_in_picture } / 3

CurPicIsI (DWord 3, bit 1)

Format:	U1

Specifies that the current picture is comprised solely of I slices and that there are no P or B slices in the picture.

Value	Name
0	Current picture has at least one P or B slice

Programming Notes
This bit should be set to "0". Note: The value of "1" setting ("Current picture is comprised solely of I slices") is REMOVED. This bit is used for hardware optimization only. There is not enough information to set this bit to "1" correctly.

ColPicIsI (DWord 3, bit 0)

HCP_PIC_STATE

		Format: U1 Specifies that the collocated picture is comprised solely of I slices and that there are no P or B slices in the picture. **Value** / **Name** 0 – Collocated picture has at least one P or B slice **Programming Notes** This bit should be set to "0". Note: The value of "1" setting ("Collocated picture is comprised solely of I slices") is REMOVED. This bit is used for hardware optimization only. There is not enough information to set this bit to "1" correctly.
4	31:28	**Reserved** Format: MBZ
	27	**CU packet structure**
	26	**strong_intra_smoothing_enable_flag** Format: U1
	25	**transquant_bypass_enable_flag** Format: Enable **Value** / **Name** / **Description** 0 – Disable – cu_transquant_bypass is not supported 1 – Enable – cu_transquant_bypass is supported
	24	**Reserved** Format: MBZ
	23	**amp_enabled_flag** Format: Enable **Value** / **Name** / **Description** 0 – Disable – Asymmetric motion partitions cannot be used in coding tree blocks. 1 – Enable – Support asymmetric motion partitions, i.e. PartMode equal to PART_2NxnU, PART_2NxnD, PART_nLx2N, or PART_nRx2N.
	22	**transform_skip_enabled_flag** Format: Enable **Value** / **Name** / **Description** 0 – Disable – transform_skip_flag is not supported in the residual coding 1 – Enable – transform_skip_flag is supported
	21	**BottomField**

HCP_PIC_STATE

Format:	U1

Value	Name	Description
0	Bottom Field	
1	Top Field	

Programming Notes
Must be zero for encoder only

20 **FieldPic**

Format:	U1

Value	Name	Description
0	Video Frame	
1	Video Field	

Programming Notes
Must be zero for encoder only.

19 **weighted_pred_flag**

Format:	U1

18 **weighted_bipred_flag**

Format:	U1

17 **tiles_enabled_flag**

Format:	U1

16 **entropy_coding_sync_enabled_flag**

Format:	U1

15 **loop_filter_across_tiles_enabled_flag**

Format:	U1

14 **Reserved**

Format:	MBZ

13 **sign_data_hiding_flag**

Format:	Enable

Value	Name	Description
0	Disable	Specifies that sign bit hiding is disabled.
1	Enable	Specifies that sign bit hiding is enabled.

Programming Notes

HCP_PIC_STATE

	Currently not supported in encoder, so must be set to 0 for encoding session.
12:10	**log2_parallel_merge_level_minus2**

Format:	U3

Value	Name	Programming Notes
[0,4]	Valid Range	The value of log2_parallel_merge_level_minus2 shall be in the range of 0 to Log2CtbSizeYCtbLog2SizeY - 2, inclusive.

Programming Notes
For encoder, always set to 0 (Intel restriction).

9	**constrained_intra_pred_flag**

Format:	U1

Programming Notes
For encoder, always set to 0 to disable this (Intel restriction).

8	**pcm_loop_filter_disable_flag**

Format:	U1

7:6	**diff_cu_qp_delta_depth (or named as max_dqp_depth)**

Format:	U2

Programming Notes
cu_qp_delta_enabled_flag = 1 and Max_DQP_Level = 0 is supported.
cu_qp_delta_enabled_flag/max_dqp_depth: 1/0: has cu qp delta. (cu depth <= max_dqp_depth) will have cu qp delta coded. Only allow QP change across LCU, no change across CU.
Must be zero for encoder

5	**cu_qp_delta_enabled_flag**

Format:	U1

Value	Name	Description
0	Disable	Does not allow QP change at CU level, the same QP is used for the entire LCU. Max_DQP_Level = 0 (i.e. diff_cu_qp_delta_depath = 0).
1	Enable	Allow QP change at CU level. MAX_DQP_level can be >0.

Programming Notes
cu_qp_delta_enabled_flag = 1 and Max_DQP_Level = 0 is supported.

4	**pcm_enabled_flag**

Format:	U1

HCP_PIC_STATE

	3	**sample_adaptive_offset_enabled_flag** Format: U1 **Programming Notes** Restriction: HW does not support SAO filtering for LCU size 16x16 Should be set to 0 if slice_sao_luma_flag==0 and slice_sao_chroma_flag==0.
	2:0	**Reserved** Format: MBZ
5	31:30	**Reserved** Format: MBZ
	23:20	**pcm_sample_bit_depth_luma_minus1** Format: U4
	19:16	**pcm_sample_bit_depth_chroma_minus1** Format: U4
	15:13	**max_transform_hierarchy_depth_inter (or named as tu_max_depth_inter)** Format: U3 Maximum TU split depths for inter blocks. **Programming Notes** For encoder, always set to 2 to allow max 2 levels of split. For more splitting, rely on CU split to match the content (Intel restriction).
	12:10	**max_transform_hierarchy_depth_intra (or named as tu_max_depth_intra)** Format: U3 Maximum TU split depth for intra blocks. **Programming Notes** For encoder, always set to 2 to allow max 2 levels of split. For more splitting, rely on CU split to match the content (Intel restriction).
	9:5	**pic_cr_qp_offset** Format: U5
	4:0	**pic_cb_qp_offset** Format: U5
6..18	31:0	**Reserved** Project: CHV, BSW Format: MBZ

HCP_PIPE_BUF_ADDR_STATE

HCP_PIPE_BUF_ADDR_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This state command provides the memory base addresses for the row store buffer and reconstructed picture output buffers required by the HCP.
This is a picture level state command and is shared by both encoding and decoding processes.

Programming Notes

All pixel surface addresses must be 4K byte aligned. There is a max of 8 Reference Picture Buffer Addresses, and all share the same third address DW in specifying 48-bit address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 2h HCP_PIPE_BUF_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 5Ch Name:

HCP_PIPE_BUF_ADDR_STATE

1..2	63:0	**Decoded Picture** Format: SplitBaseAddress4KByteAligned Frame buffer address for the final decoded picture YUV output.
3	31:0	**Decoded Picture Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
4..5	63:0	**Deblocking Filter Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the filter line buffer (read/write) used by the Deblocking Filter.
6	31:0	**Deblocking Filter Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
7..8	63:0	**Deblocking Filter Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile line buffer (read/write) used by the Deblocking Filter.
9	31:0	**Deblocking Filter Tile Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
10..11	63:0	**Deblocking Filter Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address of the tile column buffer (read/write) used by the Deblocking Filter.
12	31:0	**Deblocking Filter Tile Column Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
13..14	63:0	**Metadata Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Line buffer.
15	31:0	**Metadata Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
16..17	63:0	**Metadata Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Line buffer.

HCP_PIPE_BUF_ADDR_STATE

18	31:0	**Metadata Tile Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
19..20	63:0	**Metadata Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Metadata Tile Column buffer.
21	31:0	**Metadata Tile Column Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
22..23	63:0	**SAO Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Line buffer.
24	31:0	**SAO Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
25..26	63:0	**SAO Tile Line Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Tile Line buffer.
27	31:0	**SAO Tile Line Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
28..29	63:0	**SAO Tile Column Buffer** Format: SplitBaseAddress64ByteAligned Base address for the SAO Tile Column buffer.
30	31:0	**SAO Tile Column Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
31..32	63:0	**Current Motion Vector Temporal Buffer** Format: SplitBaseAddress64ByteAligned Base address for the Current Motion Vector Temporal buffer.
33	31:0	**Current Motion Vector Temporal Buffer Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]

HCP_PIPE_BUF_ADDR_STATE

34..35	63:0	**Reserved** Format: MBZ
36	31:0	**Reserved** Project: CHV, BSW Format: MBZ
37..52	63:0	**Reference Picture Base Address (RefAddr[0-7])** Format: SplitBaseAddress64ByteAligned[8] Base address of the reference picture buffer. **Programming Notes** Must be 4k byte aligned.
53	31:0	**Reference Picture Base Address Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
54..59	31:0	**Reserved**
60..61	63:0	**Reserved** Project: CHV, BSW
62	31:0	**Reserved**
63..65 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Format: MBZ
66..81	63:0	**Collocated Motion Vector Temporal Buffer[0-7]** Format: SplitBaseAddress64ByteAligned[8] Base address for the Collocated Motion Vector Temporal buffer. Specifies the 64 byte aligned buffer address for Collocated Motion Vector Temporal Buffer. For VP9, only index 0 is required to have a valid address.
82	31:0	**Collocated Motion Vector Temporal Buffer[0-7] Memory Address Attributes** Project: CHV, BSW Format: MemoryAddressAttributes [CHV, BSW]
83..84	63:0	**VP9 Probability Buffer Read/Write** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for VP9 Probability Buffer. Hardware reads in the probability for decode and write out the modified probability for future frames. Driver needs to program the Initial VP9 Probability for decoding the current frame. For Key Frame, it should contain the default Key Frame Probability. For non-Key Frame, it could be a default (non-Key) or one of the 8 Reference Buffers Probability. Driver must provide a valid Initial VP9 Probability buffer.

HCP_PIPE_BUF_ADDR_STATE		
85	31:0	**VP9 Probability Buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes [CHV, BSW]
86..87	63:0	**VP9 Segment ID Buffer Read/Write** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for VP9 SegmentID buffer. This should contain the writeout SegmentID from previous frame and will be used to predict SegmentID for the current frame. Hardware will write out SegmentID of the current frame in the same address for the next frame.
88	31:0	**VP9 Segment ID buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes [CHV, BSW]
89..90	63:0	**VP9 HVD Line Rowstore Buffer Read/Write** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for HVD Line Rowstore Buffer (bitstream decoder).
91	31:0	**VP9 HVD Line Rowstore buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes [CHV, BSW]
92..93	63:0	**VP9 HVD Tile Rowstore Buffer Read/Write** Format: SplitBaseAddress64ByteAligned Specifies the 64 byte aligned buffer address for HVD Tile Rowstore Buffer (bitstream decoder).
94	31:0	**VP9 HVD Tile Rowstore buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes [CHV, BSW]
95..96	63:48	**MBZ: Reserved**
97	31:0	**SAO Streamout Data Destination buffer Read/Write Memory Address Attributes** Format: MemoryAddressAttributes [CHV, BSW]

HCP_PIPE_MODE_SELECT

HCP_PIPE_MODE_SELECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.
The HCP_PIPE_MODE_SELECT command is responsible for general pipeline level configuration that would normally be set once for a single stream encode or decode and would not be modified on a frame workload basis.
This is a picture level state command and is shared by both encoding and decoding processes.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 0h HCP_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n (Excludes Dwords 0, 1). Value: 2h — Name: Value_2 — Project: CHV, BSW
1	31:24	**Reserved**

HCP_PIPE_MODE_SELECT

	23:17	**Reserved** Format: MBZ
	16:13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**Reserved** Project: CHV, BSW
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10	**Reserved** Project: CHV, BSW
	9	**Reserved** Project: CHV, BSW
	8	**Reserved** Project: CHV, BSW
	7:5	**Codec Standard Select** Value 0: HEVC Value 1: VP9
	4	**Reserved**
	3	**Pic Status/Error Report Enable** Format: Enable Value 0 (Disable): Disable status/error reporting Value 1 (Enable): Status/Error reporting is written out once per picture. The Pic Status/Error Report ID in DWord3 along with the status/error status bits are packed into one cache line and written to the Status/Error Buffer address in the HCP_PIPE_BUF_ADDR_STATE command. Must be zero for encoder mode.
	2	**Reserved** Project: CHV, BSW
	1:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31:0	**Media Soft-Reset Counter (per 1000 clocks)** Format: U32

HCP_PIPE_MODE_SELECT

		In decoder modes, this counter value specifies the number of clocks (per 1000) of GAC inactivity before a media soft-reset is applied to the HCP. If counter value is set to 0, the media soft-reset feature is disabled and no reset will occur. In encoder modes, this counter must be set to 0 to disable media soft reset. This feature is not supported for the encoder.

Value	Name	Project
0	Disable	CHV, BSW

3	31:0	**Pic Status/Error Report ID**

Format:	U32

The Pic Status/Error Report ID is a unique 32-bit unsigned integer assigned to each picture status/error output. Must be zero for encoder mode.

Value	Name	Description
0	32-bit unsigned	Unique ID Number
1	Reserved	

Programming Notes
Software must program different Status/Error Buffer addresses between pictures; otherwise the hardware might overwrite previously written data.

HCP_QM_STATE

HCP_QM_STATE

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_QM_STATE command loads the custom HEVC quantization tables into local RAM and may be issued up to 20 times: 3x Colour Component plus 2x intra/inter plus 4x SizeID minus 4 for the 32x32 chroma components.
When the scaling_list_enable_flag is set to disable, the scaling matrix is still sent to the decoder, and with all entries programmed to the same value = 16.
This is a picture level state command and is issued in both encoding and decoding processes.

Dwords 2-17 form a table for the DCT coefficients, 4 8-bit coefficients/DWord.

- Size 4x4 for SizeID0, DWords 2-5.
- Size 8x8 for SizeID1/2/3, DWords 2-17.

SizeID 0 (Table 4-10)

4x4	[31:24]	[23:16]	[15:8]	[7:0]
DWord 2	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 3	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 4	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 5	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)

SizeID 1, 2, 3 (Table 4-11)

8x8	[31:24]	[23:16]	[15:8]	[7:0]	[31:24]	[23:16]	[15:8]	[7:0]
DWord 3,2	AC(0,7)	AC(0,6)	AC(0,5)	AC(0,4)	AC(0,3)	AC(0,2)	AC(0,1)	DC
DWord 5,4	AC(1,7)	AC(1,6)	AC(1,5)	AC(1,4)	AC(1,3)	AC(1,2)	AC(1,1)	AC(1,0)
DWord 7,6	AC(2,7)	AC(2,6)	AC(2,5)	AC(2,4)	AC(2,3)	AC(2,2)	AC(2,1)	AC(2,0)
DWord 9,8	AC(3,7)	AC(3,6)	AC(3,5)	AC(3,4)	AC(3,3)	AC(3,2)	AC(3,1)	AC(3,0)
DWord 11,10	AC(4,7)	AC(4,6)	AC(4,5)	AC(4,4)	AC(4,3)	AC(4,2)	AC(4,1)	AC(4,0)
DWord 13,12	AC(5,7)	AC(5,6)	AC(5,5)	AC(5,4)	AC(5,3)	AC(5,2)	AC(5,1)	AC(5,0)
DWord 15,14	AC(6,7)	AC(6,6)	AC(6,5)	AC(6,4)	AC(6,3)	AC(6,2)	AC(6,1)	AC(6,0)
DWord 17,16	AC(7,7)	AC(7,6)	AC(7,5)	AC(7,4)	AC(7,3)	AC(7,2)	AC(7,1)	AC(7,0)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

HCP_QM_STATE

	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 4h HCP_QM_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 10h; Name:
1	31	**Reserved** Format: MBZ
	30:13	**Reserved** Format: MBZ
	12:5	**DC Coefficient** Format: U8 Specifies the 8-bit DC coefficient for SizeID 2 and 3. **Programming Notes** The DC Coefficient must be set to zero for SizeID 0 and 1. The DC Coefficient must be set to scaling_list_dc_coef_minus8 + 8 for SizeID 2 and 3.
	4:3	**Color Component** Format: U2 Encoder: When RDOQ is enabled, scaling list for all 3 color components must be same. So this field is set to always 0.

Value	Name
0	Luma
1	Chroma Cb
2	Chroma Cr
3	Reserved

HCP_QM_STATE

	2:1	**SizeID** Format: U2 **Value / Name / Description** 0 / 4x4 / 1 / 8x8 / 2 / 16x16 / 3 / 32x32 / (Illegal Value for Colour Component Chroma Cr and Cb.)
	0	**Prediction Type** Format: U1 **Value / Name** 0 / Intra 1 / Inter
2..17	511:0	**See Tables 4-10 and 4-11**

HCP_REF_IDX_STATE

HCP_REF_IDX_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a slice level command used in both encoding and decoding processes. For decoder, it is issued with the HCP_BSD_OBJECT command.

Unlike AVC, HEVC allows 16 reference idx entries in each of the L0 and L1 list for a progressive picture. Hence, a max total 32 reference idx in both lists together. The same when the picture is a field picture. Regardless the number of reference idx entries, there are only max 8 reference pictures exist at any one time. Multiple reference idx can point to the same reference picture and can optionally pic a top or bottom field, or frame.

For P-Slice, this command is issued only once, representing L0 list. For B-Slice, this command can be issued up to two times, one for L0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 12h HCP_REF_IDX_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 10h Name:

HCP_REF_IDX_STATE

1	31:5	**Reserved**
	4:1	**num_ref_idx_l[RefPicListNum]_active_minus1** Format: U4 num_ref_idx_l[*RefPicListNum*]_active_minus1 Value: [0-14]; Name:
	0	**RefPicListNum** Format: U1 Value 0: Reference Picture List 0 Value 1: Reference Picture List 1
2..17	31:16	**Reserved** Format: MBZ
	15	**list_entry_lX[i]: bottom_field_flag** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Value 0: Bottom Field Value 1: Top Field **Programming Notes** Not supported in encoder mode.
	14	**list_entry_lX[i]: field_pic_flag** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Value 0: Video Frame Value 1: Video Field **Programming Notes** Not supported in encoder mode.
	13	**list_entry_lX[i]: LongTermReference** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15.

HCP_REF_IDX_STATE

Value	Name
0	Short term reference
1	Long term reference

12	**luma_weight_IX_flag[i]** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15.

Value	Name
0	Default weighted prediction for luma
1	Explicit weighted prediction for Luma

11	**chroma_weight_IX_flag[i]** Format: U1 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15.

Value	Name
0	Default weighted prediction for Chroma
1	Explicit weighted prediction for Chroma

10:8	**list_entry_IX[i]: Reference Picture Frame ID (RefAddr[0-7])** Format: U3 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. The reference picture frame ID identifies the reference picture associated with the base address defined in **Reference Picture Address (RefAddr[0-7])** in the HCP_PIPE_BUF_ADDR_STATE command.
7:0	**list_entry_IX[i]: Reference Picture tb Value** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. clip(-128,127, CurrentPOC - RefPOC), where RefPOC is the POC value of the reference picture. 8-bit signed. See the "Derivation process for temporal luma motion vector prediction" in the HEVC standard.

HCP_SLICE_STATE

HCP_SLICE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This is a slice level command used in both encoding and decoding processes. For decoder, it is issued with the HCP_BSD_OBJECT command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 14h HCP_SLICE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 4h; Name: ; Project: CHV, BSW
1	31:25	**Reserved** Format: MBZ
	24:16	**SliceStartCtbY or (slice_start_lcu_y encoder)** Format: U9 Specifies the starting row address of the first coding tree block in the current slice.

HCP_SLICE_STATE

	15:9	**Reserved** Format: MBZ
	8:0	**SliceStartCtbX or (slice_start_lcu_x encoder)** Format: U9 Specifies the starting column address of the first coding tree block in the current slice.
2	31:25	**Reserved** Format: MBZ
	24:16	**NextSliceStartCtbY or (next_slice_start_lcu_y encoder)** Format: U9 Specifies the starting row address of the first coding tree block in the next slice. Must be set to zero when the current slice is the last slice of a picture. For the single slice per frame case, the only slice is also the last slice, so this parameter should be set to a number larger than the frame height (at least +1).
	15	**Reserved** Project: CHV, BSW
	14:9	**Reserved** Format: MBZ
	8:0	**NextSliceStartCtbX or (next_slice_start_lcu_x encoder)** Format: U9 Specifies the starting column address of the first coding tree block in the next slice. Must be set to zero when the current slice is the last slice of a picture. For the single slice per frame case, the only slice is also the last slice, so this parameter should be set to a number larger than the frame width (at least +1).
3	31:24	**Reserved** Format: MBZ
	23	**Reserved** Project: Format: MBZ
	22	**Reserved** Project: CHV, BSW
	21:17	**slice_cr_qp_offset** Format: S4 For deblocking purpose, the pic and slice level cr qp offset must be provided separately. PAK needs to perform final_chroma_cr_qp_offset = pic_cr_qp_offset + slice_cr_qp_offset.

HCP_SLICE_STATE

Value	Name
14h	-12
15h	-11
16h	-10
17h	-9
18h	-8
19h	-7
1Ah	-6
1Bh	-5
1Ch	-4
1Dh	-3
1Eh	-2
1Fh	-1
0h	0
1h	1
2h	2
3h	3
4h	4
5h	5
6h	6
7h	7
8h	8
9h	9
10h	10
11h	11
12h	12

Programming Notes
The valid value is from -12 to 12 (or 14h to 0Ch).

16:12 **slice_cb_qp_offset**

Format:	S4

For deblocking purpose, the pic and slice level cb qp offset must be provided separately.
PAK needs to perform final_chroma_cb_qp_offset = pic_cb_qp_offset + slice_cb_qp_offset.

Value	Name
14h	-12

HCP_SLICE_STATE

15h	-11
16h	-10
17h	-9
18h	-8
19h	-7
1Ah	-6
1Bh	-5
1Ch	-4
1Dh	-3
1Eh	-2
1Fh	-1
0h	0
1h	1
2h	2
3h	3
4h	4
5h	5
6h	6
7h	7
8h	8
9h	9
10h	10
11h	11
12h	12

Programming Notes
The valid value is from -12 to 12 (or 14h to 0Ch).

11:6 **SliceQp**

Format:	U6

Specifies the initial absolute value of QPy quantization parameter for the slice as defined in the Slice Header Semantics section of the HEVC standard.
This signifies only the magnitude of SliceQp. In 8 bit, SliceQp only goes from 0 to 51. But in 10 bit, it needs to go from -12 to 51. There is a sign bit specifies at bit [3] below.

5 **slice_temporal_mvp_enable_flag**

Format:	U1

HCP_SLICE_STATE

	4	**dependent_slice_flag** Format: U1 Decoder only.
	2	**LastSliceofPic** Format: U1 Value 0: Not the last slice of the picture Value 1: Last slice of the picture
	1:0	**slice_type** Format: U2 Value 0: B-slice Value 1: P-slice Value 2: I-slice Value 3: Illegal/Reserved
4	31:29	**Reserved** Format: MBZ
	28:26	**CollocatedRefIDX** Format: U3 Collocated Motion Vector Temporal Buffer Index.
	25:23	**MaxMergeIDX** Format: U3 MaxNumMergeCand = 5 - five_minus_max_num_merge_cand -1. Value 0: Name 0 Value 1: Name 1 Value 2: Name 2 Value 3: Name 3 Value 4: Name 4 **Programming Notes** The valid value is from 0 to 4 (MaxNumMergeCand = 5 - five_minus_max_num_merge_cand -1)

HCP_SLICE_STATE

	22	**cabac_init_flag** Format: U1
	21:19	**luma_log2_weight_denom** Format: U3
	18:16	**ChromaLog2WeightDenom** Format: U3
	15	**collocated_from_l0_flag** Format: U1
	14	**isLowDelay** Format: U1 If the POCs of all pictures in both lists are less than the current POC, then set to one, else set to zero.
	13	**mvd_l1_zero_flag** Format: U1 Decoder only.
	12	**slice_sao_luma_flag** Format: U1 **Programming Notes** Note: For encoder, all Slices must have same setting within a picture
	11	**slice_sao_chroma_flag** Format: U1 **Programming Notes** Note: For encoder, all Slices must have same setting within a picture
	10	**slice_loop_filter_across_slices_enabled_flag** Format: U1 **Programming Notes** Note: For the encoder, this bit must be set to zero (hardware restriction).
	9	**Reserved**
	8:5	**slice_beta_offset_div2 or (final Beta_Offset_div2 Encoder)** Format: S3 Deblocking filter beta offset. Specified in 2's comp. **Value** [1101b,0011b] — **Name** [-3,3]

HCP_SLICE_STATE

	4:1	**slice_tc_offset_div2 or (final tc_offset_div2 Encoder)** Format: S3 Deblocking filter tc offset. Specified in 2's comp. Value: [1101b,0011b] — Name: [-3,3]
	0	**slice_header_disable_deblocking_filter_flag** Format: U1
5	31:16	**Reserved** Format: MBZ
	15:0	**SliceHeaderLength** Format: U16 Decoder only. Specifies the length in bytes of the slice header including the start code. The starting byte of the slice header in the bit stream buffer is indicated by the Indirect Data Start Address in the HCP_BSD_OBJECT command. The ending byte of the slice header in the same bit stream buffer is indicated by the last byte prior to the slice data (CABAC).
6..8	31:0	**Reserved** Project: CHV, BSW Format: MBZ

HCP_SURFACE_STATE

HCP_SURFACE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

The HCP_SURFACE_STATE command is responsible for defining the frame buffer pitch and the offset of the chroma component.
This is a picture level state command and is shared by both encoding and decoding processes.
Note : Only NV12 and Tile Y are being supported for HEVC. Hence full pitch and interleaved UV is always in use. U and V Xoffset must be set to 0; U and V Yoffset must be 16-pixel aligned. This Surface State is not the same as that of the 3D engine and of the MFX pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 1h HCP_SURFACE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: U12 (Excludes Dwords 0, 1). Value: 1h — Name:
1	31:28	**Surface Id** Format: U4

HCP_SURFACE_STATE

Value	Name	Description
0h	HEVC: For current decoded Picture	8-bit uncompressed data
1h	Source Input Picture (encoder)	8-bit uncompressed data
2h	Prev Reference Picture	(VP9 only) Previous Reference
3h	Golden Reference Picture	(VP9 only) Golden Reference
4h	AltRef Reference Picture	(VP9 only) AltRef Reference
5h	Reference Picture	

27:17 **Reserved**

Format:	MBZ

16:0 **Surface Pitch Minus1**

Format:	U17-1

This field specifies the surface pitch in (#Bytes - 1).

Programming Notes

- For tiled surfaces, the pitch must be a multiple of the tile width.
- If **Half Pitch for Chroma** is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.

For Y-tiled surfaces: Range = [127, 524287]->[128B,256KB] = [1 tile, 2048 tiles].

2

27:15 **Reserved**

Format:	MBZ

14:0 **Y Offset for U(Cb) in pixel**

Format:	U15_Pixel_Row_Offset

This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled. This field is only used for PLANAR surface formats.

Programming Notes

- For PLANAR_420 surface formats, the alignment of this field follows the tile mode described in bits 14:13 of the **Memory Address Attributes** table.
- TileY (legacy 4k) - 32 pixel aligned
- TileYF (New 4k) - 64 pixel aligned
- TileYS (64k) - 256 pixel aligned

HCP_TILE_STATE

HCP_TILE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 11h HCP_TILE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: U12 (Excludes Dwords 0, 1). Value: Bh — Name:
1	31:10	**Reserved** Format: MBZ
	9:5	**NumTileColumnsMinus1** Format: U5 Specifies the number of tile columns in Ctbs per picture.

HCP_TILE_STATE		
	4:0	**NumTileRowsMinus1** Format: U5 Specifies the number of tile rows in Ctbs per picture.
2	31:24	**Ctb column position of tile column 3** Format: U8
	23:16	**Ctb column position of tile column 2** Format: U8
	15:8	**Ctb column position of tile column 1** Format: U8
	7:0	**Ctb column position of tile column 0** Format: U8
3	31:24	**Ctb column position of tile column 7** Format: U8
	23:16	**Ctb column position of tile column 6** Format: U8
	15:8	**Ctb column position of tile column 5** Format: U8
	7:0	**Ctb column position of tile column 4** Format: U8
4	31:24	**Ctb column position of tile column 11** Format: U8
	23:16	**Ctb column position of tile column 10** Format: U8
	15:8	**Ctb column position of tile column 9** Format: U8
	7:0	**Ctb column position of tile column 8** Format: U8
5	31:24	**Ctb column position of tile column 15** Format: U8
	23:16	**Ctb column position of tile column 14** Format: U8
	15:8	**Ctb column position of tile column 13** Format: U8
	7:0	**Ctb column position of tile column 12** Format: U8

HCP_TILE_STATE		
6	31:24	**Ctb column position of tile column 19** Format: U8
	23:16	**Ctb column position of tile column 18** Format: U8
	15:8	**Ctb column position of tile column 17** Format: U8
	7:0	**Ctb column position of tile column 16** Format: U8
7	31:24	**Ctb row position of tile row 3** Format: U8
	23:16	**Ctb row position of tile row 2** Format: U8
	15:8	**Ctb row position of tile row 1** Format: U8
	7:0	**Ctb row position of tile row 0** Format: U8
8	31:24	**Ctb row position of tile row 7** Format: U8
	23:16	**Ctb row position of tile row 6** Format: U8
	15:8	**Ctb row position of tile row 5** Format: U8
	7:0	**Ctb row position of tile row 4** Format: U8
9	31:24	**Ctb row position of tile row 11** Format: U8
	23:16	**Ctb row position of tile row 10** Format: U8
	15:8	**Ctb row position of tile row 9** Format: U8
	7:0	**Ctb row position of tile row 8** Format: U8
10	31:24	**Ctb row position of tile row 15** Format: U8

HCP_TILE_STATE		
	23:16	**Ctb row position of tile row 14** Format: U8
	15:8	**Ctb row position of tile row 13** Format: U8
	7:0	**Ctb row position of tile row 12** Format: U8
11	31:24	**Ctb row position of tile row 19** Format: U8
	23:16	**Ctb row position of tile row 18** Format: U8
	15:8	**Ctb row position of tile row 17** Format: U8
	7:0	**Ctb row position of tile row 16** Format: U8
12	31:16	**Reserved** Format: MBZ
	15:8	**Ctb row position of tile row 21** Format: U8
	7:0	**Ctb row position of tile row 20** Format: U8

HCP_WEIGHTOFFSET_STATE

HCP_WEIGHTOFFSET_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The HCP is selected with the **Media Instruction Opcode "7h"** for all HCP Commands. Each HCP command has assigned a media instruction command as defined in DWord 0, BitField 22:16.

This slice level command is issued in both the encoding and decoding processes, if the weighted_pred_flag or weighted_bipred_flag equals one. If zero, then this command is not issued.
Weight Prediction Values are provided in this command. Only Explicit Weight Prediction is supported in encoder.
For P-Slice, this command is issued only once together with HCP_REF_IDX_STATE Command for L0 list. For B-Slice, this command can be issued up to two times together with HCP_REF_IDX_STATE Command, one for L0 list and one for L1 list.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline Type** Default Value: 2h Format: OpCode
	26:23	**Media Instruction Opcode** Default Value: 7h Codec/Engine Name Format: OpCode Codec/Engine Name = HCP = 7h
	22:16	**Media Instruction Command** Default Value: 13h HCP_WEIGHTOFFSET_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**Dword Length** Format: =n (Excludes Dwords 0, 1). Value: 20h; Name: 32; Project:
1	31:1	**Reserved**

HCP_WEIGHTOFFSET_STATE

		Format: MBZ
	0	**RefPicListNum** Format: U1 Value / Name 0 – Reference Picture List 0 1 – Reference Picture List 1
2..17	31:16	**Reserved**
	15:8	**luma_offset_lX[i]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for luma is enabled, otherwise must be zero.
	7:0	**delta_luma_weight_lX[i]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for luma is enabled, otherwise must be zero.
18..33	31:24	**ChromaOffsetLX [i][1]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero.
	23:16	**delta_chroma_weight_lX[i][1]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero.
	15:8	**ChromaOffsetLX[i][0]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero.

HCP_WEIGHTOFFSET_STATE		
	7:0	**delta_chroma_weight_lX[i][0]** Format: U8 Where X is the RefPicListNum and i is the list entry number 0 through 15. DW2 corresponds to i=0, DW17 corresponds to i=15. Valid only if explicit weighted prediction for chroma is enabled, otherwise must be zero.

HI8DS Render Target Write MSD

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 0h Project: All Format: Opcode Full precision data message
	29	**Reserved**
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message

MSD_RTW_HI8DS - HI8DS Render Target Write MSD

13 **Reserved**

Project:	CHV, BSW
Format:	MBZ

Ignored

12 **Last Render Target Select**

Project:	All
Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 **Slot Group Select**

Project:	All
Format:	MDC_RT_SGS [CHV, BSW]

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

10:8 **Render Target Message Subtype**

Default Value:	3h
Project:	All
Format:	Opcode

SIMD8 dual source message. Use slots [15:8] for pixel enables, X/Y addresses, and oMask.

Programming Notes

The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:24] are referenced instead of [15:8].

7:0 **Binding Table Index**

Project:	All
Format:	MDC_BTS [CHV, BSW]

Specifies the Binding Table Index for the message

Illegal

illegal - Illegal

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The Illegal Opcode Exception Enable flag in cr0.1 is normally set so the normal processing of an illegal opcode is to transfer control to the System Routine. Instruction dispatch treats any unused 8-bit opcode (including bit 7 of the instruction, reserved for future opcode expansion) as if it is the illegal opcode. The illegal opcode is zero because that byte value is more likely than most to be read via a wayward instruction pointer. The illegal instruction is an instruction only in the same way that a NULL pointer in software is a pointer. Both are special values indicating invalid instances.

Format: illegal

Restriction

The illegal instruction takes no instruction options.

Syntax

illegal

Pseudocode

{ Set the Illegal Opcode Exception Status bit in cr0.1. if (Illegal Opcode Exception Enable is set in cr0.1) { Transfer control to the System Routine (return address to AIP, IP = SIP). } }

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

Integer Subtraction with Borrow

subb - Integer Subtraction with Borrow

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The subb instruction performs component-wise subtraction of src0 and src1 and stores the results in dst, it also stores the borrow into acc. If the operation produces a borrow (src0 < src1), write 0x00000001 to acc, else write 0x00000000 to acc.

Format: [(pred)] subb[.cmod] (exec_size) dst src0 src1

Restriction

AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

Syntax

[(pred)] subb[.cmod] (exec_size) reg reg reg [(pred)] subb[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] - src1.chan[n]; acc.chan[n] = borrow(src.chan[n] - src1.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Join

join - Join

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The join instruction makes the inactive channels active at the join IP if those channels are predicated. Any deactivated channels due to a goto instruction match the join IP are activated (qualified with predicates at join). If no IP is matched at this join, the program goes to the next IP with the active channels which followed the program path up to the join instruction. If no active channels are present after executing the join instruction, the program jumps to the offset specified by JIP instead of next IP. The join instruction is used in conjunction with a goto instruction. The join activates channels that are deactivated by the goto instruction. See the goto instruction for the deactivation rules. The goto and join instructions enable unstructured program control flow. These instructions must be used with additional care where dangling channels can result without proper compiler checks, meaning that it is expected that programs will navigate through these paths to reactivate the channels. Hardware does not provide native checks or reconvergence. The following table describes the 32-bit JIP. In GEN binary, JIP is at location src1 and must be of type D (signed DWord integer). JIP must be an immediate operand and is a signed 32-bit number. This value is added to IP pre-increment. If SPF is ON, none of the PcIP are updated.

Format: [(pred)] join (exec_size) JIP

Programming Notes

An index of 0 does nothing, continuing execution with the next instruction.

An index of -16 (if the jmpi instruction is in native format) or -8 (if the jmpi instruction is in compact format) is an infinite loop on the jmpi instruction.

Restriction

The {NoMask} instruction option must be specified.

The index data type must be D (Signed DWord Integer).

Syntax

[(pred)] join (exec_size) imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if (WrEn.chan[n] ) { // for the predicated channels and the remaining channels PcIP[n] = IP + 1; } } if ( PcIP != (IP + 1) ) { // for all channels when no channels are activated and no other active channels Jump(IP + JIP); }
```

Errata	Description
	A join instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW
	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Jump Indexed

jmpi - Jump Indexed

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

Description

The jmpi instruction redirects program execution to an index offset relative to the post-incremented instruction pointer. The index is a signed integer value, with positive or zero integers for forward jumps, and negative integers for backward jumps. Note: Unlike other flow control instructions, the offset used by jmpi is relative to the incremented instruction pointer rather than the IP value for the instruction itself. In GEN binary, index is at location src1. The ip register must be put (for example, by the assembler) at the dst and src0 locations. Predication is allowed to provide conditional jump with a scalar condition. As the execution size is 1, the first channel of PMASK (flags post prediction control and negate) is used to determine whether the jump is taken or not. If the condition is false, the jump is not taken and execution continues with the next instruction.

Format: [(pred)] jmpi (1) index {NoMask}

Programming Notes

An index of 0 does nothing, continuing execution with the next instruction.

An index of -16 (if the jmpi instruction is in native format) or -8 (if the jmpi instruction is in compact format) is an infinite loop on the jmpi instruction.

Restriction

The execution size must be 1.

The {NoMask} instruction option must be specified.

The index data type must be D (Signed DWord Integer).

QtrCtrl must not be used for jmpi instruction.

Syntax

[(pred)] jmpi (1) reg32 {NoMask} [(pred)] jmpi (1) imm32 {NoMask}

Pseudocode

```
Evaluate(WrEn); if ( WrEn != 0 ) { Jump(IP + 1 + index ); // IP + 1 is a pseudocode idiom for the IP of the following instruction. }
```

Errata	Description
	A jmpi instruction must not be followed by any instruction requiring register indirect access on source operands.

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

jmpi - Jump Indexed

Src Types
D

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW
	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Leading Zero Detection

lzd - Leading Zero Detection

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The lzd instruction counts component-wise the leading zeros from src0 and stores the resulting counts in dst. If src0 is zero, store 32 in dst.

Format: [(pred)] lzd[.cmod] (exec_size) dst src0

Restriction

Accumulator cannot be destination, implicit or explicit.

Syntax

[(pred)] lzd[.cmod] (exec_size) reg reg [(pred)] lzd[.cmod] (exec_size) reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { UD udScalar = src0.chan[n]; UD cnt = 0; while ( (udScalar & (1 « 31)) == 0 && cnt != 32 ) { cnt ++; udScalar = udScalar « 1; } dst.chan[n] = cnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
D, UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Line

line - Line

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

The line instruction computes a component-wise line equation (v = p * u + q where u, v are vectors and p, q are scalars) of src0 and src1 and stores the results in dst. src1 is the input vector u. src0 provides input scalars p and q, where p is the scalar value based on the region description of src0 and q is the scalar value implied from src0 region. Specifically, q is the fourth component of the 4-tuple (128-bit aligned) that p belongs to.

Format: [(pred)] line[.cmod] (exec_size) dst src0 src1

Restriction

This is a specialized instruction that only supports an execution size (ExecSize) of 8 or 16.

The src0 region must be a replicated scalar (with HorzStride == VertStride == 0).

src0 must specify .0 or .4 as the subregister number, corresponding to a subregister byte offset of 0 or 16.

Source operands cannot be accumulators.

Syntax

[(pred)] line[.cmod] (exec_size) reg reg reg [(pred)] line[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { dwP = src0.RegNum.SubRegNum[bits4:2]; // A DWord-aligned scalar. dwQ = src0.RegNum.(SubRegNum[bit4] | 0x8); // Fourth component. if ( WrEn.chan[n] ) { dst.chan[n] = dwP * src1.chan[n] + dwQ; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Linear Interpolation

lrp - Linear Interpolation

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The lrp instruction takes component-wise multiplication of src0 and src1, and adds the result to the component-wise multiplication of src2 and (1 - src0), and then stores the final results in dst.

Format: [(pred)] lrp[.cmod] (exec_size) dst src0 src1 src2

Restriction

The vertical stride (VertStride) is overloaded to 4 in HW for 3-source instructions.

The overflow conditional modifier (.o) is not allowed.

No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.

All three-source instructions have certain restrictions, described in Instruction Formats [CHV, BSW].

Syntax

[(pred)] lrp[.cmod] (exec_size) reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src1.chan[n] * src0.chan[n] + src2.chan[n] * (1.0 - src0.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	84	**Reserved** Format: MBZ

lrp - Linear Interpolation

	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved**
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved**
	34	**Reserved** Format: MBZ
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

LO8DS Render Target Write MSD

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 0h Project: All Format: Opcode Full precision data message
	29	**Reserved**
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message

MSD_RTW_LO8DS - LO8DS Render Target Write MSD

13	**Reserved** Project: CHV, BSW Format: MBZ Ignored
12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
10:8	**Render Target Message Subtype** Default Value: 2h Project: All Format: Opcode SIMD8 dual source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].
7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Logic And

and - Logic And

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The and instruction performs component-wise logic AND operation between src0 and src1 and stores the results in dst. Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a AND (NOT b) to be calculated with one instruction. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.

Format: Source modifier is not allowed if source is an accumulator.

Restriction

Source modifier is not allowed if source is an accumulator.

Syntax

[(pred)] and[.cmod] (exec_size) reg reg reg [(pred)] and[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] & src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	N	N

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*W,*D,*Q	*W,*D,*Q	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Logic Not

not - Logic Not

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

Description
The not instruction performs logical NOT operation (or one's complement) of src0 and storing the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.
A register source operand can use a source modifier: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). Such a source modifier is not particularly useful with the not instruction, as it changes the effect of not to just copying bits.
Format: [(pred)] not[.cmod] (exec_size) dst src0

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax
[(pred)] not[.cmod] (exec_size) reg reg [(pred)] not[.cmod] (exec_size) reg imm32

Pseudocode
Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { dst.chan[n] = ~ src0.chan[n]; } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*W,*D,*Q	*W,*D,*Q	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Logic Or

or - Logic Or

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

Description
The or instruction performs component-wise logic OR operation between src0 and src1 and stores the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.
Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a OR (NOT b) to be calculated with one instruction.
Format: [(pred)] or[.cmod] (exec_size) dst src0 src1

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax
[(pred)] or[.cmod] (exec_size) reg reg reg [(pred)] or[.cmod] (exec_size) reg reg imm32

Pseudocode
Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { dst.chan[n] = src0.chan[n] | src1.chan[n]; } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*W,*D,*Q	*W,*D,*Q	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Logic Xor

xor - Logic Xor

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

Description
The xor instruction performs component-wise logic XOR operation between src0 and src1 and stores the results in dst. This operation does not produce sign or overflow conditions. Only the .e/.z or .ne/.nz conditional modifiers should be used.
Register source operands can use source modifiers: Any source modifier is logical, optionally changing a source value s to ~s (inverting all source bits). This capability allows expressions like a XOR (NOT b) to be calculated with one instruction.
Format: [(pred)] xor[.cmod] (exec_size) dst src0 src1

Restriction
Source modifier is not allowed if source is an accumulator.

Syntax
[(pred)] xor[.cmod] (exec_size) reg reg reg [(pred)] xor[.cmod] (exec_size) reg reg imm32

Pseudocode
Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { dst.chan[n] = src0.chan[n] ^ src1.chan[n];}}

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	Y

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*W,*D,*Q	*W,*D,*Q	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

MEDIA_CURBE_LOAD

MEDIA_CURBE_LOAD

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Workaround: See "GPGPU Command Workarounds" section for additional programming constraints for this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_CURBE_LOAD Format: OpCode
	23:16	**SubOpcode** Default Value: 1h MEDIA_CURBE_LOAD SubOp Format: OpCode
	15:0	**DWord Length** Project: All Format: =n Total Length - 2 Value: 2h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31:0	**Reserved**
2	31:17	**Reserved**
	16:0	**CURBE Total Data Length** Project: All Format: U17 In Bytes Description: This field provides the length in bytes of the CURBE data. This field must have the same alignment as the Curbe Object Data Start Address.As the CURBE data are sent directly to ROB, range is limited to CURBE Allocation Size. This field must be 64-byte aligned.

MEDIA_CURBE_LOAD

3	31:0	**CURBE Data Start Address** Project: All Format: DynamicStateOffset[31:0] CURBE **Description** (Project): Specifies the 64-byte aligned address of the CURBE data. This pointer is relative to the **Dynamics Base Address**. (CHV, BSW) **Value** / **Name**: [0, FFFFFFFFh]

MEDIA_INTERFACE_DESCRIPTOR_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

A Media_State_Flush should be used before this command to ensure that the temporary Interface Descriptor storage is cleared.

Workaround: See "GPGPU Command Workarounds" section for additional programming constraints for this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_INTERFACE_DESCRIPTOR_LOAD Format: OpCode
	23:16	**SubOpcode** Default Value: 2h MEDIA_INTERFACE_DESCRIPTOR_LOAD SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description: 2h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:0	**Reserved**
2	31:17	**Reserved**
	16:0	**Interface Descriptor Total Length** Format: U17 In bytes This field provides the length in bytes of the Interface Descriptor data. This field must have the same alignment as the Interface Descriptor Data Start Address.It must be DQWord (32-byte) aligned. As the Interface Descriptor data are sent directly to ROB, range is limited to CURBE Allocation Size. Value / Name / Project: [32,2048] / [1,64] interface descriptor entries / CHV, BSW

MEDIA_INTERFACE_DESCRIPTOR_LOAD

3	31:0	**Interface Descriptor Data Start Address** Format: DynamicStateOffset[31:0]INTERFACE_DESCRIPTOR_DATA **Description** / **Project** This bit specifies the 64-byte aligned address of the Interface Descriptor data. This pointer is relative to the Dynamics Base Address. / CHV, BSW **Value** / **Name** [0, FFFFFFFFh] /

MEDIA_OBJECT

MEDIA_OBJECT

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT Format: OpCode
	23:16	**Media Command Sub-Opcode** Default Value: 0h MEDIA_OBJECT SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 4h DWORD_COUNT_n Project: CHV, BSW Format: =n Total Length - 2 Excludes DWords 0,1 **Generic Mode:** DWord Length = N+4, where N is in the range of [0,504]. The maximum is 504 DW (equivalent to 63 8-DW registers). When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length N and indirect data length rounded up to 8-DW aligned individually). The minimal inline data length is 0.
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Project: CHV, BSW Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.

MEDIA_OBJECT

2	31	**Children Present** Format: Enable Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. *In order avoid deadlock, such dereference must be issued once and only once for each URB handle.*
	30:27	**Reserved**
	26:25	**Reserved**
	24	**Thread Synchronization** This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message. Value 0: No thread synchronization Value 1: Thread dispatch is synchronized by the 'spawn root thread' message
	23	**Reserved**
	22	**Force Destination** Project: CHV, BSW If set, bits 20:17 are used to determine the destination of this dispatch, if clear the destination will be chosen based on load.
	21	**Use Scoreboard** This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard. Value 0: Not using scoreboard Value 1: Using scoreboard
	20:19	**Slice Destination Select** Project: CHV, BSW This bit along with the subslice destination select determines the slice that this thread must be sent to. Ignored if **Force Destination** = 0, or if product only has 1 slice. 00b: Slice 0 01b: Slice 1 — Cannot be used in products without a Slice 1. 10b: Slice 2 — Cannot be used in products without a Slice 2. 11b: Reserved

MEDIA_OBJECT

	18:17	**SubSlice Destination Select** Project: CHV, BSW This field selects the SubSlice that this thread must be sent to. Ignored if **Force Destination** = 0

Value	Name	Project
11b	Reserved	CHV, BSW
10b	SubSlice 2	
01b	SubSlice 1	
00b	SubSlice 0	

	16:0	**Indirect Data Length** Format: U17 In bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length and indirect data length rounded up to 8-DW aligned).
3	31:0	**Indirect Data Start Address** Format: GraphicsAddress[31:0]

Description
This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation.
This field specifies the 64-byte aligned address of the indirect data.

Value	Name
[0,512MB]	

Programming Notes
Bits 31:29 MBZ

4	31:25	**Reserved**
	24:16	**Scoredboard Y** Project: Format: U9 This field provides the Y term of the scoreboard value of the current thread.
	15:9	**Reserved**

MEDIA_OBJECT

	8:0	**Scoreboard X** Project: Format: U9 This field provides the X term of the scoreboard value of the current thread.
5	31:20	**Reserved** Project: Format: MBZ
	19:16	**Scoreboard Color** Project: Format: U4 This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control.
	15:8	**Reserved** Format: MBZ
	7:0	**Scoreboard Mask** Project: Format: Boolean Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE command. **Bit n (for n = 0...7):** Scoreboard n is dependent, where bit 0 maps to n = 0.
6..n	31:0	**Inline Data** Generic Mode: The format of this data is specified by software. Hardware does not interpret this data; it merely passes it to the kernel for processing. The total size for the inline data and indirect data must not exceed 112 registers.

MEDIA_OBJECT_GRPID

MEDIA_OBJECT_GRPID

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

The MEDIA_OBJECT_GRPID command is a variation of MEDIA_OBJECT which includes a group id which is used to allocate and track Barriers and Shared Local Memory. The Interface Descriptor is used to specify how much SLM is needed and how many threads will be reporting to the Barrier. All MEDIA_OBJECT_GRPIDs with the same group id should have the same interface descriptor and be dispatched to the same Tslice – the dispatcher will ensure this if Force Destination = 0, but software must ensure this if Force Destination = 1. Software should also ensure that all the threads needed for the Barrier will fit into a Tslice, or the Barrier will never be satisfied. Either SLM or a barrier must be used with MEDIA_OBJECT_GRPID, if neither is needed then a MEDIA_OBJECT must be used instead.

MEDIA_OBJECT_GRPID supports the GPGPU version of payload delivery – either indirect or CURBE can be split between the threads in a group (per-thread payload), as well as a section which is sent to all threads (cross-thread payload). See the GPGPU payload section. For indirect, the same pointer must be sent with all the commands associated with the thread group for payload splitting to work properly. Inline data is not split, but the payload attached to each command is sent with that thread. Only one of inline, indirect, or CURBE is allowed, but at least one form of payload must be sent.

MEDIA_STATE_FLUSH with the watermark bit must be placed between groups created by MEDIA_OBJECT_GRPID. The Interface Descriptor associated with the watermark must match the Interface Descriptor used for the following group.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_GRPID Format: OpCode
	23:16	**Media Command Sub-Opcode** Default Value: 6h MEDIA_OBJECT_GRPID SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 5h DWORD_COUNT_n Format: =n Total Length - 2 Excludes DWords 0,1 **Generic Mode:** DWord Length = N+5, where N is in the range of [0,504].

MEDIA_OBJECT_GRPID

		The maximum is 504 DW (equivalent to 63 8-DW registers). When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length N and indirect data length rounded up to 8-DW aligned individually). The minimal inline data length is 0.
1	31:8	**Reserved**
	7:6	**Reserved** Format: MBZ
	5:0	**Interface Descriptor Offset** Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors. **Value** [0,30] — **Name**
2	31:24	**Reserved** Format: MBZ
	23	**End of Thread Group** This bit indicates that this dispatch is the last for the current thread group.
	22	**Force Destination** Project: CHV, BSW If set, bits 20:17 are used to determine the destination of this dispatch, if clear the destination will be chosen based on load.
	21	**Use Scoreboard** Project: This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard.

Value	Name
0	Not using scoreboard
1	Using scoreboard

20:19 **Slice Destination Select**

This bit along with the Tslice destination select determines the slice that this thread must be sent to. Ignored if **Force Destination** = 0, or if product only has 1 slice.

Value	Name	Description
00b	Slice 0	
01b	Slice 1	Cannot be used in products without a Slice 1.
10b	Slice 2	Cannot be used in products without a Slice 2.
11b	Reserved	

MEDIA_OBJECT_GRPID

	18:17	**SubSlice Destination Select** Project: CHV, BSW This field selects the SubSlice that this thread must be sent to. Ignored if **Force Destination** = 0 **Value** / **Name** / **Project** 11b / Reserved / CHV, BSW 10b / SubSlice 2 / 01b / SubSlice 1 / 00b / SubSlice 0 /
	16:0	**Indirect Data Length** Format: U17 In bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than 112 (with both inline data length and indirect data length rounded up to 8-DW aligned).
3	31:0	**Indirect Data Start Address** Format: GraphicsAddress[31:0] **Description** / **Project** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. / It is the 64-byte aligned address of the indirect data. / CHV, BSW **Value** / **Name** / **Description** [0-512MB] / / Bits 31:29 MBZ
4	31:25	**Reserved** Format: MBZ
	24:16	**Scoreboard Y** Project: Format: U9 This field provides the Y term of the scoreboard value of the current thread.
	15:9	**Reserved** Format: MBZ

MEDIA_OBJECT_GRPID

	8:0	**Scoreboard X** Project: | Format: | U9 This field provides the X term of the scoreboard value of the current thread.
5	31:20	**Reserved** Project: | Format: | MBZ
	19:16	**Scoreboard Color** Project: | Format: | U4 This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control.
	15:8	**Reserved** Format: | MBZ
	7:0	**Scoreboard Mask** Project: | Format: | Boolean Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE command. **Bit n (for n = 0...7):** Scoreboard n is dependent, where bit 0 maps to n = 0.
6	31:0	**GroupID** A unique identifying number which describes the threads which share a barrier and/or SLM. Reuse of numbers is allowed as long as the old group is not currently running.
7..n	31:0	**Inline Data** The format of this data is specified by software. Hardware does not interpret this data; it merely passes it to the kernel for processing. The total size for the inline data and indirect data must not exceed 112 registers.

MEDIA_OBJECT_PRT

MEDIA_OBJECT_PRT

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

command is for generating Persistent Root Thread for the media pipeline. It only supports loading of inline data but not indirect data. This command should be used for a root thread that might have to be present in the system for a period longer than the certain minimal context-switch interrupt latency. It has to honor the context interrupt signal to terminate upon request. It should also handle replay from theinterrupted point upon context restore (the same thread being dispatched more than once). In contrary, if a thread is not a Persistent Root Thread, if dispatched, it must run to completion. The command can be used in all VFE modes, except VLD mode.

For simplification, _PRT command has a fixed size of 16 DWORD

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_PRT Format: OpCode
	23:16	**SubOpcode** Default Value: 2h MEDIA_OBJECT_PRT SubOp Format: OpCode
	15:0	**DWord Length** Project: CHV, BSW Format: =n Total Length - 2 Note: Regardless of the mode, inline data must be present in this command. The command size must fit within 16 dwords. Value: 0Eh — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31:6	**Reserved** Format: MBZ

MEDIA_OBJECT_PRT

	5:0	**Interface Descriptor Offset** Project: CHV, BSW Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31	**Children Present** Format: Enable Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. In order avoid deadlock, such de-reference must be issued once and only once for each URB handle.
	30:24	**Reserved** Format: MBZ
	23	**PRT_Fence Needed** Format: Enable This field specifies that a PRT_Fence is generated after dispatching the thread associated with this MEDIA_OBJECT_PRT. The PRT_Fence prevents additional threads following this persistent root thread until a thread spawn message is sent. The PRT_Fence is generated on first dispatch of the persistent root, as well as on re-dispatches of the persistent root after context restore.
	22	**PRT_FenceType** This field specifies the type of fence the PRT thread uses. If this field is set to 0, the fence is set at the end of the root thread queue. It will block the dispatch of the next root thread, but allowed these root threads to be populated through VFE to the root thread queue in TS. If this field is set to 1, the fence is set at the entry of VFE, similar to the fence set by the MEDIA_STATE_FLUSH command. No more command can go into the media pipe until a thread spawn message is sent (by the PRT). This field is only valid when PRT_Fence Needed is set to 1. Otherwise, it is ignored by hardware.
	21:0	**Reserved**
3	31:0	**Reserved**
4..15	31:0	**Inline Data** Format: U32

Value	Name	Description
0h	Root thread queue	Root thread queue fence
1h	VFE state flush	VFE state flush fence

MEDIA_OBJECT_WALKER

MEDIA_OBJECT_WALKER

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MEDIA_OBJECT_WALKER Format: OpCode
	23:16	**SubOpcode** Default Value: 03h MEDIA_OBJECT_WALKER SubOp Format: OpCode
	15:0	**DWord Length** Default Value: 0Fh DWORD_COUNT_n Format: =n Total Length - 2 **Note:** If this field is greater than 15, it indicates that inline data is present. If present, inline data is common for all threads generated from this command, If this field is 15, it indicates that inline data is not present. It should be noted that unlike other media object command, inline data is optional for this command.
1	31:8	**Reserved**
	7:6	**Reserved** Format: Reserved
	5:0	**Interface Descriptor Offset** Project: CHV, BSW Format: U6 This field specifies the offset from the interface descriptor base pointer to the interface descriptor which will be applied to this object. It is specified in units of interface descriptors.
2	31	**Children Present** Format: Boolean

MEDIA_OBJECT_WALKER

		Indicates that the root thread may send spawn messages to spawn child threads and/or synchronized root threads. If Children Present is not set, TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched. If Children Present is set, the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread. *In order avoid deadlock, such dereference must be issued once and only once for each URB handle.*
	30:25	**Reserved** Format: MBZ
	24	**Thread Synchronization** This field when set indicates that the dispatch of the thread originated from this command is based on the "spawn root thread" message. 0 – No thread synchronization 1 – Thread dispatch is synchronized by the 'spawn root thread' message
	23:22	**Reserved** Format: MBZ
	21	**Use Scoreboard** This field specifies whether the thread associated with this command uses hardware scoreboard. Only when this field is set, the scoreboard control fields in the VFE Dword are valid. If this field is cleared, the thread associated with this command bypasses hardware scoreboard. 0 – Not using scoreboard 1 – Using scoreboard
	20:17	**Reserved** Format: MBZ
	16:0	**Indirect Data Length** Format: U17 in bytes This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. It must be DQWord (32-byte) aligned. As the indirect data are sent directly to URB, range is limited to 496 DW. When both inline and indirect data are fetched for this command, the total size in 8-DW registers must be less than or equal to 63 (with both inline data length and indirect data length rounded up to 8-DW aligned).
3	31:0	**Indirect Data Start Address** Format: IndirectObjectOffset[31:0]

MEDIA_OBJECT_WALKER

		Description / **Project** This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the **Indirect Object Base Address.** Hardware ignores this field if indirect data is not present. Alignment of this address depends on the mode of operation. / It is the 64-byte aligned address of the indirect data / CHV, BSW **Value** / **Name** / **Description** [0 - 512MB] / / (Bits 31:29 MBZ)
4	31:0	**Reserved** Format: MBZ
5	31:8	**Group ID Loop Select** Project: CHV, BSW This bit field chooses which of the nested loops of the walker are used to identify threads which share a group id and therefore a shared barrier and SLM. The programmer must ensure that each group will fit into a single subslice. When barriers are enabled every group must have the same number of threads matching the number specified in the Interface Descriptor.

Value	Name	Description
0		Groups are not created, barriers and SLM are not allocated
1		Each complete iteration of the Color loop defines a group, the group id is the concatenation of the Outer global, Inner global, Outer local, Mid local and Inner local loop execution counts.
2		Each complete iteration of the Inner local loop and Color loop defines a group, the group id is the concatenation of the Outer global loop to the Mid local loop execution counts.
3		Each complete iteration of the Mid local loop and lower loops defines a group, the group id is the concatenation of the Outer global loop to the Outer local loop execution counts.
4		Each complete iteration of the Outer local loop and lower loops defines a group, the group id is the concatenation of the Outer global loop and the Inner global loop execution counts.
5		Each complete iteration of the Inner global loop and lower loops defines a group, the group id is the Outer global loop execution count.

	7:0	**Scoreboard Mask** Format: Boolean Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. All threads generated by this walker command share the same dynamic mask. **Bit n (for n = 0...7):** Scoreboard n is dependent, where bit 0 maps to n = 0.

MEDIA_OBJECT_WALKER		
6	31:29	**Reserved** Project: CHV, BSW
	28	**Reserved** Format: MBZ
	27:24	**Color Count Minus One** Format: U4 This field specifies the number of repeat of the inner most loop of the walker. Each repeated walk position is assigned with an incremental Color number. The Color number together with the X and Y position of the thread is used for dependency scoreboard control. **Usage Example:** This allows multiple sets of dependency threads to be dispatched.
	23:21	**Reserved** Format: MBZ
	20:16	**Middle Loop Extra Steps** Format: U5
	15:14	**Reserved** Format: MBZ
	13:12	**Local Mid-Loop Unit Y** Format: S1
	11:10	**Reserved** Format: MBZ
	9:8	**Mid-Loop Unit X** Format: S1
	7:0	**Reserved** Format: MBZ
7	31:26	**Reserved** Format: MBZ
	25:16	**Global Loop Exec Count** Format: U10
	15:10	**Reserved** Format: MBZ
	9:0	**Local Loop Exec Count** Format: U10
8	31:25	**Reserved** Format: MBZ
	24:16	**Block Resolution Y** Format: U9

MEDIA_OBJECT_WALKER

		Vertical resolution of the local loop.
	15:9	**Reserved** Format: MBZ
	8:0	**Block Resolution X** Format: U9 Horizontal resolution of the local loop.
9	31:25	**Reserved** Format: MBZ
	24:16	**Local Start Y** Format: U9 Starting vertical position of the local loop.
	15:9	**Reserved** Format: MBZ
	8:0	**Local Start X** Format: U9 Starting horizontal position of the local loop.
10	31:25	**Reserved** Format: MBZ
	24:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:9	**Reserved** Format: MBZ
	8:0	**Reserved** Project: CHV, BSW Format: MBZ
11	31:26	**Reserved** Format: MBZ
	25:16	**Local Outer Loop Stride Y** Format: S9 Vertical stride of the local outer loop, in 2's complement.
	15:10	**Reserved** Format: MBZ

MEDIA_OBJECT_WALKER

	9:0	**Local Outer Loop Stride X** Format: S9 Horizontal stride of the local outer loop, in 2's complement.
12	31:26	**Reserved** Format: MBZ
	25:16	**Local Inner Loop Unit Y** Format: S9 Vertical stride of the local inner loop, in 2's complement.
	15:10	**Reserved** Format: MBZ
	9:0	**Local Inner Loop Unit X** Format: S9 Horizontal stride of the local inner loop, in 2's complement.
13	31:25	**Reserved** Format: MBZ
	24:16	**Global Resolution Y** Format: U9 Vertical resolution of the global loop.
	15:9	**Reserved** Format: MBZ
	8:0	**Global Resolution X** Format: U9 Horizontal resolution of the global loop.
14	31:26	**Reserved** Format: MBZ
	25:16	**Global Start Y** Format: S9 Starting vertical location of the global loop, in 2's complement.
	15:10	**Reserved** Format: MBZ
	9:0	**Global Start X** Format: S9

MEDIA_OBJECT_WALKER		
		Starting horizontal location of the global loop, in 2's complement.
15	31:26	**Reserved** Format: MBZ
	25:16	**Global Outer Loop Stride Y** Format: S9 Vertical stride of the global outer loop, in 2's complement.
	15:10	**Reserved** Format: MBZ
	9:0	**Global Outer Loop Stride X** Format: S9 Horizontal stride of the global outer loop, in 2's complement.
16	31:26	**Reserved** Format: MBZ
	25:16	**Global Inner Loop Unit Y** Format: S9 Vertical stride of the global inner loop, in 2's complement.
	15:10	**Reserved** Format: MBZ
	9:0	**Global Inner Loop Unit X** Format: S9 Horizontal stride of the global inner loop, in 2's complement.
17..n	31:0	**Inline Data**

MEDIA_STATE_FLUSH

MEDIA_STATE_FLUSH

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

This command updates the Message Gateway state. In particular, it updates the state for a selected Interface Descriptor. This command can be considered same as a MI_Flush except that only media parser will get flushed instead of the entire 3D/media render pipeline. The command should be programmed prior to new Media state, curbe and/or interface descriptor commands when switching to a new context or programming new state for the same context. With this command, pipelined state change is allowed for the media pipe. It should be cautious when using this command when child_present flag in the media state is enabled. This is because that CURBE state as well as Interface Descriptor state are shared between root threads and child threads. Changing these states while child threads are generated on the fly may cause unexpected behavior. Combining with MI_ARB_ON/OFF command, it is possible to support interruptability with the following command sequence where interrupt may be allowed only when MI_ARB_ON_OFF is ON: MEDIA_STATE_FLUSH VFE_STATE // VFE will hold CS if watermark isn't met MI_ARB_OFF // There must be at least one VFE command before this one MEDIA_OBJECT .... MI_ARB_ON

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_STATE_FLUSH Format: OpCode
	23:16	**SubOpcode** Default Value: 4h MEDIA_STATE_FLUSH SubOp Format: OpCode
	15:0	**DWord Length** Project: All Format: =n Total Length - 2 Value / Name / Description 0h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:9	**Reserved** Project: All

MEDIA_STATE_FLUSH

	Format:	MBZ

8 Reserved

Project:	N, CHV, BSW

7 Flush to GO

Project:	CHV, BSW
Format:	Enable

This bit indicates that the write data out of this thread group should be flushed to the point where it is visible to following commands.

Workaround	Project
Use PIPE_CONTROL with CS stall and DC flush bits set, in place of MEDIA_STATE_FLUSH with Flush-to-GO set, to work around a preemption boundary condition.	CHV, BSW*:A0

6 Watermark Required

Project:	All

This is a single bit specifying if the MEDIA_STATE_FLUSH should stall further commands until there is enough room in a half-slice for the following thread group. The characteristics of the thread group are specified in the Interface Descriptor Offset. If set, the MEDIA_STATE_FLUSH stalls CS until there are enough threads in a half-slice, and enough SLM available in the same half-slice, and a free barrier if one is required. An Interface Descriptors can be updated after a Watermarked MEDIA_STATE_FLUSH only if it has not been used in the current context. Reusing an interface desciptor requires that this bit is clear to ensure the ID cache is reloaded. If clear, the MEDIA_STATE_FLUSH stalls CS until the TDL has dispatched the last thread, allowing the CURBE and Interface Descriptors to be updated by following commands.

Programming Notes	Project
The Interface Descriptor Offset used for the flush must be the same as that used for the GPGPU_OBJECTs. GPGPU_WALKER automatically checks the Watermark conditions before starting a thread, so this bit should not be set before GPGPU_WALKER.	CHV, BSW
If pre-emption is used, the WatermarkRequired bit must not be set.	CHV, BSW*:A0

5:0 Interface Descriptor Offset

Format:	U6

This field specifies the offset from the interface descriptor base pointer to the interface descriptor which describes what resources are required to meet the watermark.

MEDIA_VFE_STATE

MEDIA_VFE_STATE

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

A stalling PIPE_CONTROL is required before MEDIA_VFE_STATE unless the only bits that are changed are scoreboard related: Scoreboard Enable, Scoreboard Type, Scoreboard Mask, Scoreboard * Delta. For these scoreboard related states, a MEDIA_STATE_FLUSH is sufficient.

- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MEDIA_VFE_STATE Format: OpCode
	23:16	**SubOpcode** Default Value: 0h MEDIA_VFE_STATE SubOp Format: OpCode
	15:0	**DWord Length** Format: =n Total Length - 2 Value / Name / Description 07h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:10	**Scratch Space Base Pointer** Format: GeneralStateOffset[31:10] Specifies the 1k-byte aligned address offset to scratch space for use by the kernel. This pointer is relative to the **General State Base Address.**
	9:8	**Reserved** Format: MBZ
	7:4	**Stack Size**

MEDIA_VFE_STATE

Value	Name	Description
[0,11]		indicating [1KBytes, 2MBytes]

Programming Notes
Since the stack uses the upper portion of the scratch space, **Stack Size =< Per Thread Scratch Space**

	3:0	**Per Thread Scratch Space** Format: U4 Specifies the amount of scratch space allowed to be used by each thread. The driver must allocate enough contiguous scratch space, pointed to by the Scratch Space Pointer, to ensure that the maximum threads in the device each get Per Thread Scratch Space size without exceeding the driver-allocated scratch space.

Value	Name	Description	Project
[0,11]		indicating [1k bytes, 2 Mbytes]: 0 -> 1k, 1->2k, 2->4k, 3->8k ... 11->2M	CHV, BSW

2	31:16	**Reserved** Format: MBZ
	15:0	**Scratch Space Base Pointer High** Format: GeneralStateOffset[47:32] This field specifies the high 16 bits of starting address of the Scratch Space Base Pointer
3	31:16	**Maximum Number of Threads** Format: U16-1 representing thread count

Range: [0, n-1] where n = (# EUs) * (# threads/EU). See *Graphics Processing Engine* for listing of #EUs and #threads in each device.
Specifies the maximum number of simultaneous root threads allowed to be active. Used to avoid potential deadlock. If child threads are not planning on being used then this field can be set to its maximum value and there will be no thread limit beyond what is currently available in the system; the maximum value can include threads in slices that have been shut down for power reasons. For GPGPU threads the maximum value must be used.

Programming Notes
MSB will be zero due to the range limit below.

	15:8	**Number of URB Entries** Format: U8 Specifies the number of URB entries that are used by the unit.

Value	Name	Description	Project
[1,64]		[1,64] Entries	CHV, BSW

MEDIA_VFE_STATE

		Programming Notes Please note that 0 is not allowed for this field.
	7	**Reset Gateway Timer** This field controls the reset of the timestamp counter maintained in Message Gateway. **Value** / **Name** 0h – Maintaining the existing timestamp state 1h – Resetting relative timer and latching the global timestamp
	6	**Bypass Gateway Control** Project: CHV, BSW This field configures Gateway to use a simple message protocol. **Value** / **Name** 0h – Maintaining OpenGateway/ForwardMsg/CloseGateway protocol (legacy mode) 1h – Bypassing OpenGateway/CloseGateway protocol
	5:3	**Reserved** Project: CHV, BSW Format: MBZ
	2	**Reserved** Format: MBZ
	1:0	**Reserved**
4	31:8	**Reserved**
	7:4	**Reserved** Format: MBZ
	3:2	**Reserved** Project: CHV, BSW Format: MBZ
	1:0	**Slice Disable** This field disables dispatch to slices and subslices for Media and GPGPU applications. It is used to limit the amount of scratch space that needs to be allocated for a context. If a particular configuration doesn't have slice or subslice then there is no impact to disabling it. **Value** / **Name** / **Description** 00b – All subslices are enabled. 01b – Slice 2 and 1 are disabled, only Slice 0 with all subslices is enabled. 10b – Reserved 11b – Slice 2 and 1 are disabled, only Slice 0 with only subslice 0 enabled.
5	31:16	**URB Entry Allocation Size** Format: U16

MEDIA_VFE_STATE

Specifies the length of each URB entry used by the unit, in 256-bit register increments. ROB address for URB starts after CURBE Allocated region. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + Number of Interface Descriptors) must be <= (number of bytes allocated for the URB in L3CNTLREG / 32 bytes per entry). Note: Number of Interface Descriptors is 64.

If SLM is enabled for GPGPU work then the number of available entries will be 1/2 the maximum URB entries.

Programming Notes

When Inline data is used with MEDIA_OBJECT or MEDIA_OBJECT_WALKER, then the URB entry allocation size must match the Inline data size. If Indirect data is being used with MEDIA_OBJECT or GPGPU_WALKER then the allocation size must be sufficient for the Indirect data. If both Inline and Indirect are being used, then the allocation size must match the sum of the Inline and Indirect.

15:0 **CURBE Allocation Size**

Format:	U16

Specifies the total length allocated for CURBE, in 256-bit register increments. ROB address for CURBE starts at address 64. (URB Entry Allocation Size * Number of URB Entries) + CURBE Allocation Size + Interface Descriptor Entries) must be less than or equal to the number of entries in the URB as described in **Configurations**. Interface Descriptor Entries is 64

If SLM is enabled for GPGPU work then the number of available entries will be ½ the maximum URB entries.

Programming Notes

CURBE Allocation Size should be 0 for GPGPU workloads that uses indirect instead of CURBE.

6 31 **Scoreboard Enable**

Project:	

This field enables and disables the hardware scoreboard in the Media Pipeline. If this field is cleared, hardware ignores the following scoreboard state fields.

Value	Name
0h	Scoreboard disabled
1h	Scoreboard enabled

30 **Scoreboard Type**

Project:	

This field selects the type of scoreboard in use.

Value	Name
0h	Stalling Scoreboard
1h	Non-Stalling Scoreboard

29:16 **Reserved**

MEDIA_VFE_STATE

	15:8	**Reserved** Project: CHV, BSW Format: MBZ
	7:0	**Scoreboard Mask** Project: Format: Enable[8] Each bit indicates the corresponding dependency scoreboard is enabled. The scoreboard is based on the relative (X, Y) distance from the current threads' (X, Y) position. **Bit n (for n = 0…7):** Score n is enabled.
7	31:28	**Scoreboard 3 Delta Y** Project: Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	27:24	**Scoreboard 3 Delta X** Project: Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	23:20	**Scoreboard 2 Delta Y** Project: Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 2, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	19:16	**Scoreboard 2 Delta X** Project: Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 2, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.

MEDIA_VFE_STATE

	15:12	**Scoreboard 1 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 1, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	11:8	**Scoreboard 1 Delta X** Project: | Format: | S3 Relative horizontal distance of the dependent instance assigned to scoreboard 1, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	7:4	**Scoreboard 0 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 0, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	3:0	**Scoreboard 0 Delta X** Project: | Format: | S3 Relative horizontal distance of the dependent instance assigned to scoreboard 0, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
8	31:28	**Scoreboard 7 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 7, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	27:24	**Scoreboard 7 Delta X** Project: | Format: | S3

MEDIA_VFE_STATE

		Relative horizontal distance of the dependent instance assigned to scoreboard 7, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	23:20	**Scoreboard 6 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 6, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	19:16	**Scoreboard 6 Delta X** Project: | Format: | S3 Relative horizontal distance of the dependent instance assigned to scoreboard 6, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	15:12	**Scoreboard 5 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 5, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	11:8	**Scoreboard 5 Delta X** Project: | Format: | S3 Relative horizontal distance of the dependent instance assigned to scoreboard 5, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.
	7:4	**Scoreboard 4 Delta Y** Project: | Format: | S3 Relative vertical distance of the dependent instance assigned to scoreboard 4, in the form of 2's compliment.

MEDIA_VFE_STATE

		Programming Notes MBZ if scoreboard is disabled.
	3:0	**Scoreboard 4 Delta X** Project: | Format: | S3 Relative horizontal distance of the dependent instance assigned to scoreboard 4, in the form of 2's compliment. **Programming Notes** MBZ if scoreboard is disabled.

Media Block Read MSD

MSD1R_MB - Media Block Read MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Other
Group: Media Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 04h Project: All Format: Opcode Media Block Read message
	13:11	**Reserved** Project: All Format: MBZ Ignored
	10:8	**Vertical Line Stride Override** Project: All Format: MDC_VLSO [CHV, BSW] If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Media Block Write MSD

MSD1W_MB - Media Block Write MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Other
Group: Media Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 0Ah Project: All Format: Opcode Media Block Write message
	13:11	**Reserved** Project: All Format: MBZ Ignored
	10:8	**Vertical Line Stride Override** Project: All Format: MDC_VLSO [CHV, BSW] If enabled, specifies the Vertical Line Stride and Vertical Line Stride Offset override fields.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Media Transpose Read MSD

MSD1R_TT - Media Transpose Read MSD

Project: CHV, BSW
Source: DataPort 1
Length Bias: 1
Family: Other
Group: Transpose Read

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18:14	**Message Type** Default Value: 00h Project: All Format: Opcode Transpose Read message
	13:8	**Reserved** Project: All Format: MBZ Ignored
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Memory Fence MSD

MSD_MEMFENCE - Memory Fence MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Other
Group: Memory Fence

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 07h Project: All Format: Opcode Memory Fence message
	13	**Commit** Project: All Format: Enable Specifies whether control is returned to the thread only after the fence has been honored.
	12:9	**L3 Flush** The L3 Flush control is one of the following GSYNC signals. Value / Name / Description 0h / Disabled **[Default]** / The L3 caches are not flushed. **Programming Notes** If multiple caches need to be flushed, the commands need to be sent separately.
	8	**Reserved**
	7:0	**Reserved**

MFC_AVC_PAK_OBJECT

MFC_AVC_PAK_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MFC_AVC_PAK_OBJECT command is the second primitive command for the AVC Encoding Pipeline. The same command is used for both CABAC and CAVLC modes. The MV Data portion of the bitstream is loaded as indirect data object.Before issuing a MFC_AVC_PAK_OBJECT command, all AVC MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command.MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice. MFC_AVC_PAK_OBJECT command follows the MbType definition like MFD. Many fields in this command are identical to that in VME output. This is intended to reduce software converting overhead from VME to PAK. Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Length -2 Value / Name / Project: 000Ah / DWORD_COUNT_n **[Default]** / CHV, BSW

MFC_AVC_PAK_OBJECT

1	31:10	**Reserved** Format: MBZ
	9:0	**Indirect PAK-MV Data Length** This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect PAK-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size. **Value** / **Name** [0,512MB)
3..10	31:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as inline data of the MFC_AVC_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section.

MFC_JPEG_ HUFF_TABLE_STATE

MFC_JPEG_ HUFF_TABLE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image. Two Huffman tables for luma and chroma will be sent to H/W, and chroma table is used for both U and V.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_JPEG_ HUFF_TABLE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Common Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0AEh Excludes DWord (0,1) Format: =n Total Length - 2
1	31:1	**Reserved** Format: MBZ
	0	**Huff Table ID** Format: U1 Huffman table destination identifier will specify one of two destinations at the encoder into which the Huffman table must be stored.

MFC_JPEG_ HUFF_TABLE_STATE

		Value / **Name** / **Description** 0 — Huffman table 0 1 — Huffman table 1
2..13	31:0	**DC_TABLE** Format: 3Bytes: Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy 12 categories with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.
14..175	31:0	**AC_TABLE** Format: 3Bytes: Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy 162 run/size with code length and code word. Each run/size has 1-byte code length, and 2-byte code word.

MFC_JPEG_SCAN_OBJECT

MFC_JPEG_SCAN_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

Encoder Only

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_JPEG_SCAN_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 001h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:26	**Reserved** Format: MBZ
	25:0	**MCU Count** Format: U26 This field indicates the number of MCUs in the Scan. MCU Count = M_x *X* M_y The number of MCUs in a row: M_x= *(X + (H_1 *8 -1)) / (H_1 *8)* The number of MCUs in a column: M_y= *(Y + (V_1*8 -1)) / (V_1*8)X*: The number of samples per line in Y-image *Y*: The number of lines in Y-image *H1*: Horizontal sampling factor of Y-image in the Frame header *V1*: Vertical sampling factor of Y-image in the Frame header

MFC_JPEG_SCAN_OBJECT

2 — 31:25 Reserved

Format:	MBZ

24:22 Huffman AC Table

AC Huffman table destination selector specifies one of two possible AC table destinations for each Y, U, V, or R, G, B. The AC Huffman tables must have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

If AC table 0 is used for Y and AC table 1 is used for U and V, it will be set to 110b. If AC table 0 is used for R, G, and B, it will be set to 000b and so on. Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit24 (V0)	The third image component must use the AC table 0.
1XXb	Bit24 (V1)	The third image component must use the AC table 1.
X0Xb	Bit23 (U0)	The second image component must use the AC table 0.
X1Xb	Bit23 (U1)	The second image component must use the AC table 1.
XX0b	Bit22 (Y0)	The first image component must use the AC table 0.
XX1b	Bit22 (Y1)	The first image component must use the AC table 1.

Restriction
When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR,\ or YUV, **Bit22** is used for the first image component, **Bit23** is used for the second image component, and **Bit24** is used for the third image component.

21 Reserved

Format:	MBZ

20:18 Huffman DC Table

DC Huffman table destination selector specifies one of two possible DC table destinations for each Y, U, V, or R, G, B. The DC Huffman tables shall have been loaded in destination 0 and 1 by the time of issuing MFC_JPEG_HUFF_TABLE_STATE Command.

if DC table 0 is used for Y and DC table 1 is used for U and V, it will be set to 110b. If DC table 0 is used for R, G, and B, it will be set to 000b and so on. Refer to the table below for the summary of actions.

Value	Name	Description
0XXb	Bit20 (V0)	The third image component must use the DC table 0.
1XXb	Bit20 (V1)	The third image component must use the DC table 1.
X0Xb	Bit19 (U0)	The second image component must use the DC table 0.
X1Xb	Bit19 (U1)	The second image component must use the DC table 1.
XX0b	Bit18 (Y0)	The first image component must use the DC table 0.
XX1b	Bit18 (Y1)	The first image component must use the DC table 1.

MFC_JPEG_SCAN_OBJECT

Restriction
When InputSurfaceFormatYUV = RGB, because the order of input image components can be RGB, GBR, BGR, YUV, **Bit18** is used for the first image component, **Bit19** is used for the second image component, and **Bit20** is used for the third image component.

17 **Head Present Flag**

If this flag is set to 0, then no MFC_JPEG_PAK_INSERT_OBJECT commands will be sent. If this flag is set to 1, then one or more MFC_JPEG_PAK_INSERT_OBJECT commands will be sent after MFC_JPEG_SCAN_OBJECT command.

Value	Name	Description
0		No insertion into the output bitstream buffer before Scan encoded bitstream
1		Headers, tables, App data insertion into the output bitstream buffer. HW will insert the insertion data before the Scan encoded bitstream.

16 **Is Last Scan**

If this flag is set, then HW will insert EOI (0xFFD9) to the end of Scan encoded bitstream.

Value	Name	Description
0		Not the last Scan.
1		Indicates that the current Scan is the last one.

15:0 **Restart Interval**

Format:	U16

Specifies the number of MCUs in an ECS, except for the last ECS. Restart maker is inserted periodically and it separates the two neighboring ECSs.

Value	Name
0-FFFFh	

Programming Notes
A value of '0' implies that the Scan Data has a single ECS.

MFC_MPEG2_PAK_OBJECT

MFC_MPEG2_PAK_OBJECT

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The MFC_MPEG2_PAK_OBJECT command is the second primitive command for the MPEG-2 Encoding Pipeline. Different from AVC, the MV Data portion of the bitstream is loaded as part of MB control data.

Before issuing a MFC_MPEG2_PAK_OBJECT command, all MPEG2_MFX states need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of this command.

MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the Start_MB_In_Slice loaded at the beginning of each slice.

MFC_ MPEG2_PAK_OBJECT command follows the MbType definition like MFD. Many fields in this command are identical to that in VME output. This is intended to reduce software converting overhead from VME to PAK.

Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_AVC_PAK_INSERT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h ENC Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h MEDIA_ Format: OpCode
	15:12	**Reserved**
	11:0	**DWord Length** Default Value: 0007h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

MFC_MPEG2_PAK_OBJECT		
1..8	31:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as inline data of the MFC_MPEG2_PAK_OBJECT command. It has a fixed size of 8 DWs. Its definition is described in the next section

MFC_MPEG2_SLICEGROUP_STATE

MFC_MPEG2_SLICEGROUP_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a slice group level command and can be issued multiple times within a picture that is comprised of multiple slice groups. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_SLICEGROUP_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2 Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h MEDIA_ Format: OpCode
	20:16	**SubOpcode B** Default Value: 3h MEDIA_ Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 6h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31	**MbRateCtrlFlag- RateControlCounterEnable (Encoder-only)** To enable the accumulation of bit allocation for rate controlThis field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields.Note: To reset MB level rate control (QRC), we need to set both bits MbRateCtrlFlag and MbRateCtrlReset to 1 in the new slice

MFC_MPEG2_SLICEGROUP_STATE

Value	Name	Project
0h	Disable	All
1h	Enable	All

30 **MbRateCtrlReset- ResetRateControlCounter (Encoder-only)**

To reset the bit allocation accumulation counter to 0 to restart the rate control.

Value	Name	Description	Project
0h	Disable	Not reset	All
1h	Enable	reset	All

29:28 **MbRateCtrlMode- RC Triggle Mode (Encoder-only)**

Value	Name	Description
00b		Always Rate Control, whereas RC becomes active if sum_act > sum_target or sum_act < sum_target
01b		Gentle Rate Control, whereas RC becomes active if sum_act > upper_midpt or sum_act < lower_midpt
10b		Loose Rate Control, whereas RC becomes active if sum_act > sum_max or sum_act < sum_min
11b		Reserved

27:24 **MbRateCtrlParam- RC Stable Tolerance (Encoder-only)**

Format:	U4

This field specifies the tolerance required to deactivate RC once it has been triggered.

Value	Name
[0, 15]	

23 **RateCtrlPanicFlag - RC Panic Enable (Encoder-only)**

If this field is set to 1, RC enters panic mode when sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.

Value	Name	Project
0	Disable	All
1	Enable	All

22 **RateCtrlPanicType - RC Panic Type (Encoder-only)**

This field selects between two RC Panic methods. If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

Value	Name	Description
0h		QP Panic
1h		CBP Panic

21 **Reserved**

MFC_MPEG2_SLICEGROUP_STATE

20	**SkipConvDisabled - MB Type Skip Conversion Disable (Encoder-only)** This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Section 2.3.3.1.6

Value	Name	Description	Project
0h	Enable	Enable skip type conversion	All
1h	Disable	Disable skip type conversion	All

19	**IsLastSliceGrp** IsLastSliceGrp = 1 if the current slice group is the last slice group of a picture; 0 otherwise.It is used by the zero filling in the Minimum Frame Size test.
18	**BitstreamOutputFlag - Compressed BitStream Output Disable Flag (Encoder-only)**

Value	Name	Description
0h	Enable	enable the writing of the output compressed bitstream
1h	Disable	disable the writing of the output compressed bitstream

17 **HeaderPresentFlag - Header Insertion Present in Bitstream (Encoder-only)**

Value	Name	Description	Project
0h	Disable	no header insertion into the output bitstream buffer, in front of the current slice encoded bits	All
1h	Enable	header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.	All

16 **SliceData PresentFlag - SliceData Insertion Present in Bitstream (Encoder-only)**

Value	Name	Description	Project
0h	Disable	no Slice Data insertion into the output bitstream buffer	All
1h	Enable	Slice Data insertion into the output bitstream buffer is present.	All

15 **TailPresentFlag - Tail Insertion Present in bitstream (Encoder-only)**

Value	Name	Description
0h		no tail insertion into the output bitstream buffer, after the current slice encoded bits
1h		tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.

14	**FirstSliceHdrDisabled** when this is on, the first slice header of the slice group is expected to be provided by the user via insertion command. PAK HW will skip it.
13	**IntraSlice** intra slice value included in slice headers, when IntraSliceFlag = 1.
12	**IntraSliceFlag** intra slice flag included in slice headers
11:8	**Reserved**

Format:	MBZ for SliceID extension

MFC_MPEG2_SLICEGROUP_STATE

	7:4	**SliceID[3:0] (Encoder-only)** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP
	3:2	**Reserved** Format: MBZ for StreamID extension
	1:0	**StreamID[1:0] (Encoder-only)** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP
2	31:24	**NextSgMbYcnt - also NextStartVertPos** Vertical count of the first MB in the next slice group (Encoder-only)Note: This field restricts total number of MB in the Y direction to 255 or less.
	23:16	**NextSgMbXcnt - also NextStartHorzPos** BitFieldDesc
	15:8	**FirstMbYcnt - also CurrStartVertPos** Project: All Format: U8 also CurrStartVertPos, Vertical count of the first MB in the current slice group (Encoder-only)
	7:0	**FirstMbXcnt - also CurrStartHorzPos** Project: All Format: U8 Horizontal count of the first MB in the current slice group (Encoder-only)
3	31:9	**Reserved** Format: MBZ
	8	**SliceGroupSkip** Project: All Exists If: //Encoder Only Format: U1 All macroblocks are skipped
	7:6	**Reserved** Format: MBZ
	5:0	**SliceGroupQp** Project: All Exists If: //Encoder Only Format: U6 Initial slice quality parameter

MFC_MPEG2_SLICEGROUP_STATE

4	31:29	**Reserved** Format: MBZ
	28:0	**BitstreamOffset - Indirect PAK-BSE Data Start Address (Write)** Exists If: //Encoder Only This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address.It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes.For Write, there is no need to have a data length field. It is assumed the global memory bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access.This field is only valid for AVC encode mode. Value: [0,512MB) — Name:
5	31:24	**MaxQpNegModifier - Magnitude of QP Max Negative Modifier (Encoder-only)** Format: U8 This field specifies the lower limit of the QP modifier. Value: [0, 51] — Name:
	23:16	**MaxQpPosModifier - Magnitude of QP Max Positive Modifier (Encoder-only)** Format: U8 This field specifies the upper limit of the QP modifier. Value: [0, 51] — Name:
	15:12	**ShrinkParam - Shrink Resistance (Encoder-only)** Format: U4 This field specifies the additional points added each time decreased correction is invoked. Value: [0, 15] — Name:
	11:8	**Shrinkaram - Shrink Init (Encoder-only)** Format: U4 This field specifies the initial points required to trip decreased control. Value: [0, 15] — Name:
	7:4	**GrowParam - Grow Resistance (Encoder-only)** Format: U4 This field specifies the additional points added each time increased correction is invoked. Value: [0, 15] — Name:

MFC_MPEG2_SLICEGROUP_STATE

	3:0	**GrowParam - Grow Init (Encoder-only)** Format: U4 This field specifies the initial points required to trip increased control. Value: [0, 15] — Name:
6	31:24	**Reserved** Format: MBZ
	23:20	**CorrectPoints - Correct 6 (Encoder-only)** Format: U4 This field specifies the points used in the lowermost RC region when sum_act <= sum_min. Value: [0, 15] — Name:
	19:16	**CorrectPoints - Correct 5 (Encoder-only)** Format: U4 This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt. Value: [0, 15] — Name:
	15:12	**CorrectPoints - Correct 4 (Encoder-only)** Format: U4 This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target. Value: [0, 15] — Name:
	11:8	**CorrectPoints - Correct 3 (Encoder-only)** Format: U4 This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt. Value: [0, 15] — Name:
	7:4	**CorrectPoints - Correct 2 (Encoder-only)** Format: U4 This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max. Value: [0, 15] — Name:
	3:0	**CorrectPoints - Correct 1 (Encoder-only)** Format: U4

MFC_MPEG2_SLICEGROUP_STATE

This field specifies the points used in the topmost RC region when sum_act > sum_max

Value	Name
[0, 15]	

7	31:28	**CV7 - Clamp Value 7 (Encoder-only)** Exists If: //Encoder Only
	27:24	**CV6 - Clamp Value 6 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	23:20	**CV5 - Clamp Value 5 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	19:16	**CV4 - Clamp Value 4 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	15:12	**CV3 - Clamp Value 3 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	11:8	**CV2 - Clamp Value 2 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	7:4	**CV1 - Clamp Value 1 (Encoder-only)** Project: All Exists If: //Encoder Only Format: U4
	3:0	**CV0 - Clamp Value 0 (Encoder-only)** If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped). For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following: none | none | CV7 | CV6 | CV5 | CV4 | CV3 | CV3

MFC_MPEG2_SLICEGROUP_STATE

none	CV7	CV6	CV5	CV4	CV3	CV3	CV2
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV2
CV6	CV5	CV4	CV3	CV3	CV2	CV2	CV1
CV5	CV4	CV3	CV3	CV2	CV2	CV1	CV1
CV4	CV3	CV3	CV2	CV2	CV1	CV1	CV0
CV3	CV3	CV2	CV2	CV1	CV1	CV0	CV0
CV3	CV2	CV2	CV1	CV1	CV0	CV0	CV0

For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV6	CV5	CV4	CV3	CV2	CV1
none	CV7	CV6	CV5	CV4	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV2	CV2	CV1
CV6	CV5	CV4	CV4	CV3	CV2	CV1	CV0
CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0

MFD_AVC_BSD_OBJECT

MFD_AVC_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

Description
The MFD_AVC_BSD_OBJECT command is the only primitive command for the AVC Decoding Pipeline. The same command is used for both CABAC and CAVLD modes. The Slice Data portion of the bitstream is loaded as indirect data object.Before issuing a MFD_AVC_BSD_OBJECT command, all AVC states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_AVC_BSD_OBJECT command.
Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_AVC_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 4h — Name: Excludes DWord (0,1) = 0004 **[Default]** — Project: CHV, BSW

MFD_AVC_BSD_OBJECT

1	31:0	**Indirect BSD Data Length** Format: U32 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address. AVC Short Format : It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect BSD Data Start Address** Project: CHV, BSW Format: U29 This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the **MFD Indirect Object Base Address**. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes. In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0. It includes the NAL Header (the NAL Header does not need to perform EMU detection). For AVC, it is a single byte. But for MVC, the NAL Header is 4 Bytes long. These NAL Header Unit must be passed to HW in the compressed bitstream buffer. **Value** / **Name** [0,512MB)
3..5	31:0	**Inline Data** All the required Slice Header parameters and error handling settings are captured as InLine Data of the AVC_BSD_OBJECT command. It has a fixed size of 4 DWs. Its definition is described in the follwoing section: Inline Data Description [CHV, BSW].

MFD_AVC_DPB_STATE

MFD_AVC_DPB_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a frame level state command used only in DXVA2 AVC Short Slice Bitstream Format VLD mode. RefFrameList[16] of DXVA2 interface is replaced with intel Reference Picture Addresses[16] of MFX_PIPE_BUF_ADDR_STATE command. The LongTerm Picture flag indicator of all reference pictures are collected into LongTermPic_Flag[16]. FieldOrderCntList[16][2] and CurrFieldOrderCnt[2] of DXVA2 interface are replaced with intel POCList[34] of MFX_AVC_DIRECTMODE_STATE command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 9h — Name: Excludes DWord (0,1) **[Default]** — Project: CHV, BSW

MFD_AVC_DPB_STATE

1	31:16	**LongTermFrame_Flag[16][1 bit]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame. **Value** / **Name** 1 — the picture is a long term reference picture 0 — the picture is a short term reference picture
	15:0	**Non-ExistingFrame_Flag[16][1 bit]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 1 bit per reference frame. **Value** / **Name** / **Description** 1 / INVALID / the reference picture in that entry of RefFrameList[] does not exist anymore. 0 / VALID / the reference picture in that entry of RefFrameList[] is a valid reference **Programming Notes** When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the corresponding bit of NonExistingFrameFlags shall be set to 0.
2	31:0	**UsedForReference_Flag[16][2 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 2 bits per reference frame. **Value** / **Name** / **Description** 0 / NOT_REFERENCE / indicates a frame is "not used for reference". 1 / TOP_FIELD / bit[0] indicates that the top field of a frame is marked as "used for reference". 2 / BOTTOM_FIELD / bit[1] indicates that the bottom field of a frame is marked as "used for reference". 3 / FRAME / bit[1:0] indicates that a frame (or field pair) is marked as "used for reference".
3..10	31:0	**LTSTFrameNumList[16][16 bits]** One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame.Depending on the corresponding LongTermFrame_Flag[], the content of this field is interpreted differently. **Value** / **Name** / **Description** 1 / LongTermFrame_Flag[i] / LTSTFrameNumList[i] represent LongTermFrameIdx. 0 / ShortTermFrame_Flag[i] / LTSTFrameNumList[i]represent Short Term Picture FrameNum. **Programming Notes** When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.

MFD_AVC_DPB_STATE

11..18 **Project:** CHV, BSW	31:0	**ViewIDList[16][16 bits]** Project: CHV, BSW One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. The view ids are 10-bits, the upper 6 bits are ignored."000000" & ViewId1[9:0] & "000000" & ViewId0[9:0] **Programming Notes** When an Intel RefFrameList[i] is not an valid entries, Viewid should be set to 0x00
19..22 **Project:** CHV, BSW	31:0	**ViewOrderListL0[16][8 bits]** Project: CHV, BSW One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored. 0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL0[i] is not an valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.
23..26 **Project:** CHV, BSW	31:0	**ViewOrderListL1[16][8 bits]** Project: CHV, BSW One-to-one correspondence with the entries of the Intel RefFrameList[16]. 8 bits per reference frame. The view order need 4-bits, the upper 4 bits are ignored. 0000 & ViewOrder3[3:0] & 0000 & ViewOrder2[3:0] & 0000 & ViewOrder1[3:0] & 0000 & ViewOrder0[3:0] **Programming Notes** When the ViewOrderListL1[i] is not an valid inter-view reference, its corresponding ViewOrder should be set to 0xF Since only interview with the same polarity will be used, there is no need to have field bit in this list. Hardware is going to append correct polarity bit as needed.

MFD_AVC_PICID_STATE

MFD_AVC_PICID_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a frame level state command used for both AVC Long and Short Format in VLD mode.PictureID[16] contains the pictureID of each reference picture (16 maximum) so hardware can uniquely identify the reference picture across frames (this will be used for DMV operation).This command will be needed for both short and long format.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h MFD_AVC_DPB_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h DEC Format: OpCode
	20:16	**SubOpcode B** Default Value: 5h MEDIA_ Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0008h Excludes DWord (0,1) Project: CHV, BSW Format: =n Total Length - 2
1	31:1	**Reserved** Project: All Format: MBZ

MFD_AVC_PICID_STATE

	0	**PictureID Remapping Disable** Project: CHV, BSW **Value** / **Name** / **Description** / **Project** 0h — AVC decoder will use 16 bits Picture ID to handle DMV and identify the reference picture — Desc — All 1h — AVC decoder will use 4 bits FrameStoreID (index to RefFrameList) to handle DMV and identify the reference picture — Desc — All **Programming Notes** If Picture ID Remapping Disable is "1", PictureIDList will not be used.
2..9	31:0	**PictureIDList[16][16 bits]** Project: CHV, BSW One-to-one correspondence with the entries of the Intel RefFrameList[16]. 16 bits per reference frame. PictureID of each Frame uniquely identifies the reference picture across frames. The same number cannot be reused until the referece picture is completely retired(no longer used for referece)When an element of the list of frames is not relevant (e.g., due to the corresponding reference entry being empty or being marked as "not used for reference"), the value of the LTSTFrameNumList entry shall be set to 0.

MFD_AVC_SLICEADDR

MFD_AVC_SLICEADDR

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a Slice level command used only for DXVA2 AVC Short Slice Bitstream Format VLD mode.When decoding a slice, H/W needs to know the last MB of the slice has reached in order to start decoding the next slice. It also needs to know if a slice is terminated but the last MB has not reached, error conealment should be invoked to generate those missing MBs. For AVC DXVA2 Short Format, the only way to know the last MB position of the current slice, H/W needs to snoop into the next slice's start MB address (a linear address encoded in the Slice Header). Since each BSD Object command can have only one indirect bitstream buffer address, this command is added to help H/W to snoop into the next slice's slice header and retrieve its Start MB Address. This command will take the next slice's bitstream buffer address as input (exactly the same way as a BSD Object command), and parse only the first_mb_in_slice syntax element. The result will stored inside the H/W, and will be used to decode the current slice specified in the BSD Object command.Only the very first few bytes (max 5 bytes for a max 4K picture) of the Slice Header will be decoded, the rest of the bitstream are don't care. This is because the first_mb_in_slice is encoded in Exponential Golomb, and will take 33 bits to represent the max 256 x 256 = 64K-1 value. The indirect data of MFD_AVC_SLICEADDR is a valid BSD object and is decoded as in BSD OBJECT command.The next Slice Start MB Address is also exposed to the MMIO interface.The Slice Start MB Address (first_mb_in_slice) is a linear MB address count; but it is translated into the corresponding 2D MB X and Y raster position, and are stored internally as NextSliceMbY and NextSliceMbX.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_AVC_ SLICEADDR Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 7h Format: OpCode
	15:12	**Reserved** Format: MBZ

MFD_AVC_SLICEADDR

	11:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Project** 1h / Excludes DWord (0,1) **[Default]** / CHV, BSW
1	31:0	**Indirect BSD Data Length** Project: CHV, BSW Format: U32 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. Driver always programs this up to 5 bytes; for bitstream less than 5 bytes, driver program the lesser value. (Emulation Prevention Byte should never happen for the first 5 bytes when the max picture size can only be 4Kx4K)It is the length in bytes of the bitstream data for the current slice, including Slice Header + Slice Data + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly in both CABAC and CAVLC modes.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect BSD Data Start Address** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLD Modes.In implementing a phantom slice at the end of a picture for automatic error concealment, this field should set to 0.It includes the NAL Header Byte. (but does not perform EMU detection).Must provide a valid MB address, even if error. MB must be clamped to within a pic boundary. **Value** / **Name** [0,512MB) /

MFD_IT_OBJECT

MFD_IT_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

All weight mode (default and implicit) are mapped to explicit mode. But the weights come in either as explicit or implicit.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_IT_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 06h Excludes DWord (0,1) For AVC = Ch Format: =n Total Length - 2 Note: Regardless of the mode, inline data must be present in this command.
1	31:10	**Reserved** Format: MBZ
	9:0	**Indirect IT-MV Data Length** Format: U10 FormatDesc: In bytes This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-MV Data Start Address field is ignored.

MFD_IT_OBJECT

		This field must have the same alignment as the Indirect Object Data Start Address.AVC-IT Mode: It must be DWord aligned (since each MV is 4bytes in size)Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data).
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect IT-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into the IT pipeline for processing. This pointer is relative to the Indirect IT-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect MV Data Length is set to 0. Alignment of this address depends on the mode of operation.AVC-IT Mode: It must be DWord aligned (since each MV is 4 bytes in size). This field is only valid in AVC decoder IT mode (VC1 and MPEG uses inline MV data). Value: [0,512MB) — Name:
3	31:12	**Reserved** Format: MBZ
	11:0	**Indirect IT-COEFF Data Length** Project: All This field provides the length in bytes of the indirect data, which contains all the non-zero coefficients for the current MB. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-COEFF Data Start Address field is ignored. Since each IT-COEFF data is 1 DW in size, with 12 bits, this field can be extended to support up to 4:4:4 format.(256 pixel * 3 byte pixel components * 4 bytes per coeff).This field must be integer multiple of 16-bytes for AVC (since each coefficient is 4 bytes in size).This field is only valid in AVC, VC1, MPEG2 decoder IT mode. Value: [0,3072] — Name: In bytes [0, 256*3*4]
4	31:29	**Reserved**
	28:0	**Indirect IT-COEFF Data Start Address Offset** Project: All This field specifies the memory starting address (offset) of the coeff data to be loaded into the IT pipeline for processing. This pointer is relative to the Indirect IT-COEFF Object Base Address.Hardware ignores this field if indirect IT-COEFF data is not present, i.e. the Indirect IT-COEFF Data Length is set to 0.This field must be DW aligned, since each coeff icient is 4 bytes in size.Driver will determine the Num of EOB 4x4/8x8 must match the block cbp flags, if not match, hardware cannot hang - add error handling.This field is only valid in AVC, VC1, MPEG2 decoder IT mode. Value: [0,512MB) — Name:

MFD_IT_OBJECT		
5	31:6	**Reserved** Format: MBZ
	5:0	**Indirect IT-DBLK Control Data Length** Project: All Format: U6 This field provides the length in bytes of the indirect data, which contains all the deblocker control information for the current MB (in 4x4 sub-block partitioning). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect IT-DBLK Data Start Address field is ignored. This field must have the same alignment as the Indirect IT-DBLK Data Start Address. It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size.This field is only valid in AVC decoder IT mode.
6	31:29	**Reserved** Format: MBZ
	28:0	**Indirect IT-DBLK Control Data Start Address Offset** Format: IndirectObjectBaseAddress[28:0] This field specifies the memory starting address (offset) of the Deblocker control data to be fetched into the IT Pipeline for processing. This pointer is relative to the Indirect IT-DBLK Object Base Address. Hardware ignores this field if indirect data is not present, ie. The indirect IT-DBLK Control Data Length is set to 0. It must be DWord aligned. Each Deblock Control Data record is 48 bytes or 12 DWords in size. This field is only valid in AVC decoder IT mode. **Value** / **Name** [0,512MB)
7..n	31:0	**Inline Data** Union for all 3 codecs Includes IT, MC, IntraPred inline data as well as Deblocker control information AVC-IT Modes: Hardware interprets this data in the specified format. VC1-IT Modes: Hardware interprets this data in the specified format. MV inline MPEG2-IT Modes: Hardware interprets this data in the specified format. (IS mode) MV inline For AVC there 7 DWords of inline data, hence N is equal to 13.

MFD_JPEG_BSD_OBJECT

MFD_JPEG_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2
Exists If: //Decoder

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_JPEG_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 004h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:0	**Indirect Data Length** Project: All . It is the length in bytes of the bitstream data for the current Scan. It includes the first byte of the first MCU and the last non-zero byte of the last MCU in the Scan. Specifically, the zero-padding bytes (if present) are excluded. Hardware ignores the contents after the last non-zero byte.

MFD_JPEG_BSD_OBJECT

2 | 31:29 | Reserved

Project:	All
Format:	MBZ

2 | 28:0 | Indirect Data Start Address

Project:	All
Format:	IndirectObjectOffset[28:0]

This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the JPEG bitstream data

3 | 31:29 | Reserved

3 | 28:16 | Scan Horizontal Position

Project:	All
Format:	U13 bits in blocks

This field indicates the horizontal position (in block units) of the first MCU in the Scan.

3 | 15:13 | Reserved

3 | 12:0 | Scan Vertical Position

Project:	All
Format:	U13 bits in blocks

This field indicates the vertical position (in block units) of the first MCU in the Scan.

4 | 31 | Reserved

Format:	MBZ

4 | 30 | Interleaved

Value	Name	Description
0	Non-Interleaved	one component in the Scan
1	Interleaved	multiple components in the Scan

4 | 29:27 | Scan Components

Bit0: Y Bit1: U Bit2: V For example, if non-interleaved Y, then it will be set to 001b. If interleaved Y, U, and V, it will be set to 111b.

4 | 26 | Reserved

Format:	MBZ

4 | 25:0 | MCU Count

Project:	All
Format:	U26

This field indicates the number of MCUs in the Scan.

MFD_JPEG_BSD_OBJECT

5	31:16	**Reserved** Project: All Format: MBZ
	15:0	**RestartInterval(16 bit)** Project: All Format: U16 Specifies the number of MCU in restart interval. Valid values are 1->0xFFFFValue of 0 implies that all the SCAN have only one ECS.

MFD_MPEG2_BSD_OBJECT

MFD_MPEG2_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

Different from AVC and VC1, MFD_MPEG2_BSD_OBJECT command is pipelinable. This is for performance purpose as in MPEG2 a slice is defined as a group of MBs of any size that must be within a macroblock row.Slice header parameters are passed in as inline data and the bitstream data for the slice is passed in as indirect data. Of the inline data, slice_horizontal_position and slice_vertical_position determines the location within the destination picture of the first macroblock in the slice. The content in this command is identical to that in the MEDIA_OBJECT command in VLD mode described in the Media Chapter.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_MPEG2_BSD_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

MFD_MPEG2_BSD_OBJECT

1 — 31:0 — Indirect BSD Data Length

Project:	All
Format:	U32

It is the length in bytes of the bitstream data for the current slice. It includes the first byte of the first macroblock and the last non-zero byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. This field is sized to support beyond MPEG-2 MP@HL bitstream (<4K). According to Table 8-6 of ISO/IEC 13818-2, the maximum number of bits per macroblock for 4:2:0 is 4608. So the maximum slice size for 4K x 4K is 4608 * 256 / 8 = 147,456 bytes (0x24000), which requires 18 bits.

Programming Notes	Project
As MPEG-2 spec does not post any limitation of the size of zero-padding bytes, it is possible to have a slice data with large length (including zero-padding bytes). As the data beyond 0x10E00 would only be zero bytes for a valid slice data	
zero-padding restriction is removed	CHV, BSW

2 — 31:29 — Reserved

Project:	All
Format:	MBZ

2 — 28:0 — Indirect Data Start Address

Format:	IndirectObjectOffset[28:0]

This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the BSD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the MPEG2 VLD bitstream data This address points to the first byte of the MB layer data, i.e. not including slice header.

3..4 — 31:0 — Inline Data

All the required Slice Header parameters and error handling settings are captured as MPEG2_BSD_OBJECT Inline Data Descriptor structures. It has a fixed size of 2 DWs. Its definition is described in the next section.

MFD_VC1_BSD_OBJECT

MFD_VC1_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MFD_VC1_BSD_OBJECT command is the only primitive command for the VC1 Decoding Pipeline. The macroblock data portion of the bitstream is loaded as indirect data object.Before issuing a MFD_VC1_BSD_OBJECT command, all VC1 states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VC1_BSD_OBJECT command.VC1 deblock filter kernel cross the slice boundary if in the last MB row of a slice, so need to know the last MB row of a slice to disable the edge mask. There is why VC1 BSD hardware need to know the end of MB address for the current slice. As such no more phantom slice is needed for VC1, as long as the driver will program both start MB address in the current slice and the start MB address of the next slice. As a result, we can also support multiple picture state commands in between slices.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

MFD_VC1_BSD_OBJECT

1	31:24	**Reserved** Project: All Format: MBZ
	23:0	**Indirect BSD Data Length** Project: All Format: U24 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.GEN6 Long Format : It is the length in bytes of the bitstream data for the current slice/picture. It includes the first byte of the first macroblock and the last byte of the last macroblock in the slice/picture. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte (trailing zeros). This field is sized to support VC1 AP@L4 Level bitstream. It includes the byte that contains the First MB Bit OffsetGEN7 Short Format : It is the length in bytes of the bitstream data for the current slice, including Picture/Slice Header + Emulation Prevention Bytes + any filling trailing zeros after the last MB. Hardware ignores the contents after the last non-zero byte. Trailing zero is allowed and handled correctly.
2	31:29	**Reserved**
	28:0	**Indirect Data Start Address** Project: All Format: IndirectObjectOffset[28:0] This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This pointer is relative to the MFD Indirect Object Base Address.Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VC1 bitstream data. Value: [0,512MB) — Name:
3	31:24	**Reserved**
	23:16	**Slice Start Vertical Position** This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. For SecondField this value is reset to zero as oppoed to the VC1 spec Ref: 9.1.2 Slice Layer.This field is for both Long and Short VC1 Interface Format.
	15:9	**Reserved** Project: All Format: MBZ
	8:0	**Next Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks.This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering)This field is maintained and provided by the driver for both Long and Short VC1 Interface Format.

MFD_VC1_BSD_OBJECT

4 | 31:16 — First_MB_Byte_Offset of Slice Data or Slice Header

For DXVA2 VC1 Short Format onlyIt gives the byte offset to locate the first MB data in the bitstream for a slice, relative to the Indirect BSD Data Start Address.

15:5 — Reserved

Project:	All
Format:	MBZ

4 — Emulation Prevention Byte Present

Value	Name	Description	Project
0h		H/W needs to perform Emulation Byte Removal	All
1h		H/W does not need to perform Emulation Byte Removal	All

3 — Reserved

Project:	All
Format:	MBZ

2:0 — FirstMbBitOffset (First Macroblock Bit Offset)

Format:	U3

This field provides the bit offset of the first macroblock of the Slice in the first byte of the input compressed bitstream.It is used with First_MB_Byte_Offset for non-byte aligned position.

MFD_VC1_LONG_PIC_STATE

MFD_VC1_LONG_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

MFX_VC1_LONG PIC_STATE command encapsulates the decoding parameters that are read or derived from bitstream syntax elements above (inclusive) picture header layer. These parameters are static for a picture and when slice structure is present, these parameters are not changed from slice to slice of the same picture. Hence, this command is only issued at the beginning of processing a new picture and prior to the VC1_*_OBJECT command. The values set for these state variables are retained internally across slices.Only the parameters needed by hardware (BSD unit) to decode bit sequence for the macroblocks in a picture layer or a slice layer are presented in this command. Other parameters such as the ones used for inverse transform or motion compensation are provided in MFX_VC1_PRED_PIPE_STATE command.This Long interface format is intel proprietary interface. Driver will need to perform addition operations to generate all the fields in this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_LONG_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0004h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

MFD_VC1_LONG_PIC_STATE

1 — 31:24 — Reserved

Project:	CHV, BSW
Format:	MBZ

23:16 — PictureHeightInMBsMinus1 (Picture Height Minus 1 in Macroblocks)

Project:	CHV, BSW
Format:	U8

This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).This field is used in VLD and IT modes.

Value	Name	Description	Project
[0,254]	Value_0_to_254	a valid range of [0,254] [1, 255] MB	CHV, BSW

Programming Notes

Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boudary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.

15:8 — Reserved

Project:	CHV, BSW
Format:	MBZ

7:0 — PictureWidthInMBsMinus1 (Picture Width Minus 1 in Macroblocks)

Project:	CHV, BSW
Format:	U8-1

This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16).This field is used in VLD and IT modes

Value	Name	Description	Project
[0,254]	Value_0_to_254	[1,255] MB	CHV, BSW

2 — 31:24 — Bitplane Buffer Pitch Minus 1

Project:	CHV, BSW
Format:	U7-1 Pitch in (Bytes - 1).

Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format (Gen6 and Gen7), it is written by an application and later read by the HW. But in VC1 Short Format (Gen7 only), it is written and read by H/W only.This field is specified for better performance

Value	Name
[0, FFFFFFFFh]	

MFD_VC1_LONG_PIC_STATE

		Programming Notes For Gen6 : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Long Format : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row.This field is not used in IT mode, used in VLD mode only.For VC1 DXVA2 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.
	23:16	**Reserved** Project: CHV, BSW Format: MBZ
	15	**DmvSurfaceValid** Project: CHV, BSW Indicated when the DMV read surface is valid. This surface stored the direct motion vectors and Mb type.This field is set for B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture).Whne the current picture being decoded is an I, P or BI, this bit is set to 0, since there is no DMV read in these picture decoding process.This field is not used in IT mode, used in VLD mode only.
	14	**ImplicitQuantizer** Project: CHV, BSW Derived by driver from QUANTIZER.This field is used in intel VC1 VLD Long Format only, not used in IT and DXVA2 VC1.This bit is set to 1 when syntax element QUANTIZER=0, else its set to 0
	13	**Interpolation Rounder Contro** Project: CHV, BSW Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process.This field is used in VLD and IT modes. **Programming Notes** This bit field is taken from bRcontrol in DXVA_PictureParameters data structure
	12	**SyncMarker** Project: CHV, BSW Indicates whether sync markers are enabled/disabled. If enable, sync markers "may be" present in the current video sequence being decoded. It is a sequence level syntax element and is valid only for Simple and Main Profiles.

Value	Name	Description	Project
0h	Not Present	Sync Marker is not present in the bitstream	CHV, BSW
1h	Maybe present	Sync Marker maybe present in the bitstream	CHV, BSW

MFD_VC1_LONG_PIC_STATE

Programming Notes
This field is only valid in VLD mode.For Simple Profile, SyncMarker must set to 0.For Main Profile, SyncMarker can be set to 0 or 1.This field is used in both intel and MS DXVA2 VLD interface, but not used in IT mode.

11:8 **Motion Vector Mode**

Project:	CHV, BSW

This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision.Before the polarity of Chroma Half-pel or Q-pel is reversed from DXVA2 Spec, now I have fixed it to match with DXVA2 VC1 Spec.

Value	Name	Description	Project
0XX0b		Chroma Quarter -pel + Luma bicubic. (can only be 1MV)	CHV, BSW
0XX1b		Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV)	CHV, BSW
1XX0b		Chroma Quarter -pel + Luma bilinear. (can only be 1MV)	CHV, BSW
1XX1b		Chroma Half-pel + Luma bilinear	CHV, BSW

Programming Notes
Bits 11:8 are taken from bMVprecisionAndChromaRelation in DXVA_PictureParameters data structure.Bit 11 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 8 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion.This field is used in both VLD and IT modes.

7 **RangeReductionScale**

Project:	CHV, BSW

This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled.

Value	Name	Description	Project
0h		Scale down reference picture by factor of 2	CHV, BSW
1h		Scale up reference picture by factor of 2	CHV, BSW

Programming Notes
This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

MFD_VC1_LONG_PIC_STATE

6 **RangeReduction Enable**

Project:	CHV, BSW

This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element (DXVA_PictureParameters bPicDeblocked bit 5) in the Picture Header.

Value	Name	Description	Project
0h	Disable	Range reduction is not performed	All
1h	Enable	Range reduction is performed	All

Programming Notes

This field is for Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture.RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

5 **LOOPFILTER Enable Flag**

This filed is the decoded syntax element LOOPFILTER in bitstream. It indicates if In-loop Deblocking is ON according to picture level bitstream syntax control. This bit affects BSD unit and also the loop filter unit.When this bit is set to 1, PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command must also be set to 1. In this case, in-loop deblocking operation follows the VC1 standard - deblocking doesn't cross slice boundary.When this bit is set to 0, but PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command is set to 1. It indicates the loop filter unit is used for out-of-loop deblocking. In this case, deblocking operation does cross slice boundary.This field is used in VLD mode only, not in IT mode.

Value	Name	Description	Project
0h	Disable	Disables loop filter	All
1h	Enable	Enables loop filter	All

4 **Overlap Smoothing Enable Flag**

This field is the decoded syntax element OVERLAP in bitstreamIndicates if Overlap smoothing is ON at the picture levelThis field is used in both VLD and IT modes.

Value	Name	Description	Project
0h	Disable	to disable overlap smoothing filter	All
1h	Enable	to enable overlap smoothing filter	All

3 **Secondfield**

This flag is set for the second field in field pictures.This field is used in both VLD and IT modes.

2:1 **Reserved**

MFD_VC1_LONG_PIC_STATE

	0	**VC1 Profile** specifies the bitstream profile.This field is used in both VLD and IT modes.

Value	Name	Description	Project
0h	Disable	current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile)	All
1h	Enable	current picture is in Advanced Profile	All

Programming Notes
This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not.

3	31	**Reserved**
	30:29	**CondOver** This field is the decoded syntax element CONDOVER in a bitstream of advanced profile. It controls the overlap smoothing filter operation for an I frame or an BI frame when the picture level qualization step size PQUANT is 8 or lower.This field is used in intel VC1 VLD mode only, not in DXVA2 VC1 and IT modes.

Value	Name	Description
00b		No overlap smoothing
01b		Reserved
10b		Always perform overlap smoothing filter
11b		Overlap smoothing on a per macroblock basis based on OVERFLAGS

	28:26	**PicType (Picture Type)** This field specifies the coding type of the picture according to the Frame Coding Mode. When FCM = 00 | 01 (a Progressive or Interlaced Frame Picture):000 = I001 = P010 = B011 = BI100 = SkippedOther encodings are reservedWhen FCM = 10 | 11 (a Field Picture)000 = I/I001 = I/P010 = P/I011 = P/P100 = B/B101 = B/BI110 = BI/B111 = BI/BIAlthough, for a field picture, it is set for a field-pair, but HW will only look at one field state only, and the other field state is don't care. This field is read and qualified with the SecondField flag internally.This field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For DXVA2 VC1 IT mode, driver needs to convert the DXVA2 interface to intel HW VLD Long Format interface.
	25:24	**FCM (Frame Coding Mode)** This is the same as the variable FCM defined in VC1.This field must be set to 0 for Simple and Main ProfilesThis field is unique to intel VC1 VLD Long format, and is used in IT mode as well. For DXVA2 VC1 IT mode, driver needs to convert the DXVA2 interface to intel HW VLD Long Format interface.

Value	Name	Description	Project
00b	Disable	Progressive Frame Picture	All
01b	Enable	Interlaced Frame Picture	All
10b		Field Picture with Top Field First	
11b		Field Picture with Bottom Field First	

MFD_VC1_LONG_PIC_STATE

	23:21	**Reserved** Project: All Format: MBZ
	20:16	**AltPQuant (Alternative Picture Quantization Value)** This field is identical to the variable ALTPQUANT which is derived from VOPDQUANT configuration in the VC1 standard.This field must be set to 0 for Simple/Main I and BI pictures as VOPDQUANT is not present.This field is used in intel VC1 VLD Long Format mode only, not used in DXVA2 VC1 VLD and IT modes.
	15:13	**Reserved** Project: All Format: MBZ
	12:8	**PQuant (Picture Quantization Value)** Project: All Format: U5 This is the same as the calculated variable PQUANT in VC1 standard where PQuant = PQINDEX, except when QUANTIZER = 0 and PQINDEX > 8, it is given asPQuant = (PQINDEX < 29) ? PQINDEX - 3 : PQINDEX*2 - 31This field is used in all picture types (I, P, B and BI) and all operating modes (IT mode and intel and DXVA2 VLD modes).
	7:0	**BScaleFactor** BScaleFactorThis field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1 standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21 encodings of BFRACTION as shown in the table here. Other values are reserved.MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine Motion Prediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "BScaleFactor >= 128".This field is only valid for B pictures. This field is used only in intel VC1 VLD Long format mode, it is not used in DXVA2 VC1 VLD and IT modes.BFRACTION VLCBFRACTIONBScaleFactor0001/21280011/3850102/31700111/4641003/41921011/5511102/510211100003/515311100014/520411100101/64311100115/621511101001/73711101012/77411101103/711111101114/714811110005/718511110016/722211110101/83211110113/89611111005/816011111017/8224
4	31:30	**Reserved** Format: MBZ
	29:28	**UnifiedMvMode (Unified Motion Vector Mode)** This field is a combination of the variables MVMODE and MVMODE2 in the VC1 standard, for parsing Luma MVD from the bitstream. This field is used to signal 1MV vs 4MVallowed (Mixed Mode). This field is also used to signal Q-pel or Half-pel MVD read from the bitstream. The bicubic or bilinear Luma MC interpolation mode is duplicate information from Motion Vector Mode field, and is ignored here.This field is used in intel VC1 VLD Long Format mode only, it is not used in DXVA2 VC1 VLD and IT modes.

MFD_VC1_LONG_PIC_STATE

Value	Name	Description	Project
00b		Mixed MV, Q-pel bicubic	All
01b		1-MV, Q-pel bicubic	All
10b		1-MV half-pel bicubic	
11b		1-MV half-pel bilinear	

27 **FourMvSwitch (Four Motion Vector Switch)**

This field indicates if 4-MV is present for an interlaced frame P picture. It is identical to the variable 4MVSWITCH (4 Motion Vector Switch) in VC1 standard.This field is used in intel VC1 VLD Long Format mode only, it is not used in DXVA2 VC1 VLD and IT modes.

Value	Name	Description	Project
0h	Disable	only 1-MV	All
1h	Enable	1, 2, or 4 MVs	All

26 **FastUVMCFlag (Fast UV Motion Compensation Flag)**

This field specifies whether the motion vectors for UV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard.This field is used in both VLD and IT modes.It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector Mode LSBit.

Value	Name	Description
0h		no rounding
1h		quarter-pel offsets to half/full pel positions

25 **RefFieldPicPolarity (Reference Field Picture Polarity)**

This field specifies the polarity of the one reference field picture used for a field P picture. It is derived from the variable REFFIELD defined in VC1 standard and is only valid when one field is referenced (NUMREF = 0) for a field P picture.When NUMREF = 0 and REFFIELD = 0, this field is the polarity of the reference I/P field that is temporally closest; When NUMREF = 0 and REFFIELD = 1, this field is the polarity of the reference I/P field that is the second most temporally closest. The distance is measured based on display order but ignoring the repeated field if present (due to RFF = 1).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description	Project
0h		Top (even) field	All
1h		Bottom (odd) field	All

24 **NumRef (Number of References)**

This field indicates how many reference fields are referenced by the current (field) picture. It is identical to the variable NUMREF in the VC1 standard. This field is only valid for field P picture (FCM = 10 | 11).This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		One field referenced
1h		Two fields referenced

MFD_VC1_LONG_PIC_STATE

23:20	**BwdRefDist (Reference Distance)** This field is valid only in B field pictures giving the value of BRFD. The field is ignored in P Picture.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
19:16	**FwdRefDist (Reference Distance)** Format: U4 This field is the number of frames between the current frame and its reference frame. It is derived from the syntax element REFDIST (P Reference Distance) in the VC1 standard. 0 means that the previous frame is the reference frame.It has the same value as of FRFD for both P and B field pictures.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes. Value: [0, 15] — Name:
15:12	**Reserved** Format: MBZ
11:10	**ExtendedDMVRange (Extended Differential Motion Vector Range Flag)** This field specifies the differential motion vector range in interlaced pictures. It is equivalent to the variable DMVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
9:8	**ExtendedMVRange (Extended Motion Vector Range Flag)** This field specifies the motion vector range in quarter-pel or half-pel modes. It is equivalent to the variable MVRANGE in the VC1 standard. This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes
7:4	**AltPQuantEdgeMask (Alternative Picture Quantization Edge Mask)** This field is a bit mask for the four edges in clock-wise order, indicating whether AltPQuant is used for the edge macroblocks.It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found..This field is valid only if AltPQuantConfig is 01. Bit 0: Left picture edge macroblocksBit 1: Top picture edge macroblocksBit 2: Right picture edge macroblocksBit 3: Bottom picture edge macroblocksThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
00b		No extended range
01b		Extended horizontally
10b		Extended vertically
11b		Extended in both directions

Value	Name	Description
00b		[-256, 255] x [-128, 127]
01b		512, 511] x [-256, 255]
10b		[-2048, 2047] x [-1024, 1023]
11b		[-4096, 4095] x [-2048, 2047]

MFD_VC1_LONG_PIC_STATE

	3:2	**AltPQuantConfig (Alternative Picture Quantization Configuration)** This field specifies the way AltPQuant is used in the picture. It determines how to compute the macroblock quantizer step size, MQUANT. It is derived based on the following variables DQUANT, DQUANTFRM, DQPROFILE, DQSBEDGE, DQDBEDGE, and DQBILEVEL defined in the VC1 standard, as shown in Error! Reference source not found..This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description	Project
00b		AltPQuant not used	All
01b		AltPQuant is used and applied to edge macroblocks only	All
10b		MQUANT is encoded in macroblock layer	
11b		AltPQuant and PQuant are selected on macroblock basis	

	1	**HalfQP** This field is used for inverse quantization of AC coefficients. It is valid only when PQuant is used.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	0	**PQuantUniform** Indicating if uniform quantization applies to the picture. It is used for inverse quantization of the AC coefficients.QUANTIZER 001123PQUANTIZER - -01--PQINDEX>=9<=8----PQuantUniform010201ImplicitQuantizer = 0, and PQuantUniform = 0 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=0; and 2) QUANTIZER = 10b.ImplicitQuantizer = 0, and PQuantUniform = 1 is used to represent 2 cases : 1) QUANTIZER=01 and PQUANTIZER=1; and 2) QUANTIZER = 11bThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description	Project
0h		Non-uniform	All
1h		Uniform	All

5	31	**BitplanePresentFlag (Bitplane Buffer Present Flag)** This field indicates whether the bitplane buffer is present for the picture. If set, at least one of the fields listed in bits 22:16 is coded in non-raw mode, and Bitplane Buffer Base Address field in the VC1_BSD_BUF_BASE_STATE command points to the bitplane buffer. Otherwise, all the fields that are applicable for the current picture in bits 22:16 must be coded in raw mode.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		bitplane buffer is not present
1h		bitplane buffer is present

	30	**ForwardMbRaw** This field indicates whether the FORWARDMB field is coded in raw or non-raw mode.This field is only valid when PictureType is B.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

MFD_VC1_LONG_PIC_STATE

Value	Name	Description
0h		non-raw mode
1h		raw mode

29 **MvTypeMbRaw**

This field indicates whether the MVTYPREMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

28 **SkipMbRaw**

This field indicates whether the SKIPMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P or B.0 = non-raw mode1 = raw modeThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

27 **DirectMbRaw**

This field indicates whether the DIRECTMB field is coded in raw or non-raw mode.This field is only valid when PictureType is P or B.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

26 **OverflagsRaw**

This field indicates whether the OVERFLAGS field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		Non-Raw Mode
1h		Raw Mode

25 **AcPredRaw**

This field indicates whether the ACPRED field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h	Disable	Non-Raw Mode
1h	Enable	Raw Mode

MFD_VC1_LONG_PIC_STATE

24	**FieldTxRaw** This field indicates whether the FIELDTX field is coded in raw or non-raw mode.This field is only valid when PictureType is I or BI.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes. **Value** / **Name** / **Description** 0h / Disable / Non-Raw Mode 1h / Enable / Raw Mode
23	**Reserved** Project: All Format: MBZ
22:20	**MvTab (Motion Vector Table)** Project: All Format: U3 This field specifies which motion vector table(s) is (are) used for motion vector (differential) decoding in a P or B picture. This field is the combination of the variables MVTAB and IMVTAB in the VC1 standard. Two bits are defined for progressive frame pictures; And two or three bits are defined for interlaced field/frame pictures depending on NUMREF and P or B picture types.This field is valid for P and B pictures. It is not valid for I pictures.For P or B progressive frame pictures0 = Motion Vector Differential VLD Table 01 = Motion Vector Differential VLD Table 12 = Motion Vector Differential VLD Table 23 = Motion Vector Differential VLD Table 3The other encodings are reservedFor P interlace field pictures with NUMREF = 0 or P/B interlace frame pictures0 = 1-Reference Table 01 = 1-Reference Table 12 = 1-Reference Table 23 = 1-Reference Table 3The other encodings are reservedFor P interlace field picture with NUMREF = 1 or B interlaced field pictures0 = 2-Reference Table 01 = 2-Reference Table 12 = 2-Reference Table 23 = 2-Reference Table 34 = 2-Reference Table 45 = 2-Reference Table 56 = 2-Reference Table 67 = 2-Reference Table 7The other encodings are reservedThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
19:18	**FourMvBpTab (4-MV Block Pattern Table)** This field specifies which table is used to decode the 4-MV block pattern (4MVBP) syntax element in 4-MV macroblocks. It is identical to the variables 4MVBPTAB in the VC1 standard, section 9.1.1.37. This field is valid only in interlace frame P, B pictures, or interlace field P, B pictures. It is not valid for I picture.For interlace field P and B pictures, it is only valid if UnifiedMvMode is equal to Mixed-MV Type. For interlace frame P picture, it is only valid if FourMvSwitch is 1.For interlace frame B picture, it is always valid.0 = 4MVBP Table 01 = 4MVBP Table 12 = 4MVBP Table 23 = 4MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
17:16	**TwoMvBpTab (2MV Block Pattern Table)** This field specifies which table is used to decode the 2MV block pattern (2MVBP) syntax element in 2MV field macroblocks. It is identical to the variables 2MVBPTAB in the VC1 standard, section 9.1.1.36. This field is valid only in interlace frame P/B pictures. It is not valid for I picture, nor for interlace field P or B pictures.0 = 2MVBP Table 01 = 2MVBP Table 12 = 2MVBP Table 23 = 2MVBP Table 3This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

MFD_VC1_LONG_PIC_STATE

	15:14	**Reserved** Project: All Format: MBZ
	13:12	**TransType (Picture-level Transform Type)** Project: All Format: U2 This field specifies the Transform Type at picture level. It is identical to the variable TTFRM in the VC1 standard, section 7.1.1.41.This field is only valid when TransTypeMbFlag is 1. Otherwise, it is reserved and MBZ.This field is set to 00 when VSTRANSFORM is 0 in the entry point layer.00 = 8x8 Transform01 = 8x4 Transform10 = 4x8 Transform11 = 4x4 TransformThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	11	**TransTypeMbFlag (Macroblock Transform Type Flag)** This field indicates whether Transform Type is fixed at picture level or variable at macroblock level. It is identical to the variable TTMBF in the VC1 standard, section 7.1.1.40.This field is set to 1 when VSTRANSFORM is 0 in the entry point layer.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	10:8	**MbModeTab (Macroblock Mode Table)** This field signals which code table is used to decode the macroblock mode syntax element (MBMODE) in the macroblock layer in a P or B picture. This field is identical to the variables MBMODETAB in the VC1 standard, section 9.1.1.33. This field is valid for interlace frame P, B picture and interlace field P, B picture. It is not valid for I picture, nor progressive frame P, B pictures.Two bits are defined for interlace frame P, B pictures; And three bits are defined for interlaced field P, B pictures.Two bits are defined for interlace frame P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to 4-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 3Other encodings are invalidThree bits are defined for interlace field P, B pictures. There are two set of code tables selected based on if UnifiedMvMode is equal to Mixed-MV Type or not. 0 = Code Table 01 = Code Table 12 = Code Table 23 = Code Table 34 = Code Table 45 = Code Table 56 = Code Table 67 = Code Table 7This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.
	7:6	**TransAcY (Picture-level Transform Luma AC Coding Set Index, TRANSACTABLE2** BitFieldDesc
	5:4	**TransAcUV (Picture-level Transform Chroma AC Coding Set Index, TRANSACTABLE)** This field, together with PQINDEX, specifies which intra AC coding set to be used for decoding the non-zero AC coefficients in a coded luma (Y) block. This field is the combination of the variables TRANSACFRM and TRANSACFRM2 in the VC1 standard.For I pictures, TransAcY is the same as TRANSACFRM2. For other pictures, it is the same as TRANSACFRM, and therefore must be programmed to be the same as TransAcUV. This field is valid for all picture types.0 = Coding set index 01 = Coding set index 12 = Coding set index 23 is invalidThis field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description
0h		variable transform type in macroblock layer
1h		use picture level transform type TransType

MFD_VC1_LONG_PIC_STATE

	3	**TransDcTab (Intra Transform DC Table)** This field specifies whether the low motion tables or the high motion tables are used to decode the Transform DC coefficients in intra-coded blocks. This field is identical to the variable TRANSDCTAB in the VC1 standard, section 8.1.1.2.This field is valid for all picture types.This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

Value	Name	Description	Project
0h		The high motion tables	All
1h		The low motion tables	All

	2:0	**CbpTab (Coded Block Pattern Table)** This field specifies the table used to decode the CBPCY syntax element for each coded macroblock in P and B pictures. This field is combination of the variable CBPTAB for P and B frame pictures and the variable ICBPTAB in interlace field P, B pictures and interlace frame P, B pictures in the VC1 standard (Table 52 and Table 102).This field is reserved and MBZ for I or BI pictures as I only has a fixed table.000 = Table 0 (Table 169 for P, B frames or Table 124 otherwise)001 = Table 1 (Table 170 for P, B frames or Table 125 otherwise)010 = Table 2 (Table 171 for P, B frames or Table 126 otherwise)011 = Table 3 (Table 172 for P, B frames or Table 127 otherwise)100 = Table 4 (Table 128 for interlace field/frame P, B pictures)101 = Table 5 (Table 129 for interlace field/frame P, B pictures)110 = Table 6 (Table 130 for interlace field/frame P, B pictures)111 = Table 7 (Table 131 for interlace field/frame P, B pictures)This field is unique to intel VC1 VLD Long format mode, and is not used in IT and DXVA2 VC1 modes.

MFD_VC1_SHORT_PIC_STATE

MFD_VC1_SHORT_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VC1_SHORT_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_DEC Format: OpCode
	23:21	**SubOpcode A** Default Value: 1h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0003h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:24	**Reserved** Project: All Format: MBZ
	23:16	**Picture Height** Format: U8-1 Picture Height in Macroblocks This field indicates the height of the picture in unit of macroblocks. For example, for a 1920x1080

MFD_VC1_SHORT_PIC_STATE

frame picture, PictureHeightInMBs equals 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).This field is used in VLD and IT modes.Note: Even though the Advanced Profile allows frame dimensions (width, height) to not be aligned to macroblock boudary, it doesn't affect the bitstream decoding. And it is preferable to use 'intermediate buffer' that is macroblock aligned for decoding. In order to simplify the out-of-bound reference pixel access, the out-of-bound extrapolation rule in VC1 spec can be used to expand the expected decoded frame to the intermediate buffer dimension.

Value	Name	Description	Project
[0,127]	Value_0_to_127	[1, 128] MB	CHV, BSW

15:8 Reserved

Project:	All
Format:	MBZ

7:0 Picture Width

Format:	U8-1 Picture Width in Macroblocks

This field indicates the width of the picture in unit of macroblocks. For example, for a 1920x1080 frame picture, PictureWidthInMBs equals 120 (1920 divided by 16).This field is used in VLD and IT modes.

Value	Name	Description	Project
[0,127]	Value_0_to_127	[1, 128] MB	CHV, BSW

2 31:24 Bitplane Buffer Pitch Minus 1

Format:	U7-1 Pitch in Bytes

Specifies the bitplane buffer pitch in (#Bytes - 1). Bitplane buffer is a linear buffer. It is needed only when the bitplane is not encoded as raw, and therefore is present in the header explicitly. In VC1 Long Format (Gen6 and Gen7), it is written by an application and later read by the HW. In VC1 Long Format (Gen6 and Gen7), it is written by an application, and later read by the HW. But in VC1 Short Format (Gen7 only), it is written and read by H/W only.This field is specified for better performanceFor Gen6 : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Long Format : The pitch must be equal to PictureWidthInMBs/2.For Gen7 VC1 Short Format : If Pic Width is less than or equal to 2K pixels, bitplane pitch is set to 64 (one cacheline; programmed as 63) bytes per MB row. If Pic Width is greater than 2K pixels, bitplane pitch is set to 128 (two cachelines; programmed as 127) bytes per MB row.This field is not used in IT mode, used in VLD mode only.For VC1 DXVA2 Short Format, the bitplane specification is between H/W and Driver only. For Long Format, application is responsible for allocation with the driver.

23 Interpolation Rounder Control

Used only in MC operation. This field specifies the rounding control value used in interpolation operation of motion prediction process. Note: This bit field is taken from bRcontrol in DXVA_PictureParameters data structure This field is used in VLD and IT modes.

22:20 Reserved

Project:	All
Format:	MBZ

MFD_VC1_SHORT_PIC_STATE

	19:16	**Motion Vector Mode** This field indicates one of the following motion compensation interpolation modes for P and B pictures. The MC interpolation modes apply to prediction values of luminance blocks and are always in quarter-sample. For chrominance blocks, it always performs bilinear interpolation with either half-pel or quarter-pel precision.0XX0 = Chroma Quarter -pel + Luma bicubic. (can only be 1MV)0XX1 = Chroma Half-pel + Luma bicubic. (can be 1MV or 4MV)1XX0 = Chroma Quarter -pel + Luma bilinear. (can only be 1MV)1XX1 = Chroma Half-pel + Luma bilinearNote: Bits 19:16 are taken from bMVprecisionAndChromaRelation in DXVA_PictureParameters data structure.Bit 19 of Motion Vector Mode = 1 for Luma Bilinear MC; = 0 for Luma Bicubic MCBit 16 of Motion Vector Mode = 1 for half-sample Chroma motion = 0 for quarter-sample Chroma motion.This field is used in both VLD and IT modes.
	15	**DmvSurfaceValid** Indicated when the DMV read surface is valid. This surface stored the direct motion vectors. This field is set fo B pictures that can refer to a previous P picture for DMV. If there is an I-picture before a B (in decoding order) then this field is not set (as a result, zero's DMV's will be assumed while decoding the B picture. That is, there is no explicit DMV buffer for an I-picture). This field is not used in IT mode, used in VLD mode only.
	14:12	**Reserved**
	11	**VC1 Profile** Project: All specifies the bitstream profile. Note: This is required because 128 is added for intra blocks post inverse transform in advanced profile and also to find out if Motion vectors are adjusted or not. This field is used in both VLD and IT modes. Value / Name / Description 0h / **[Default]** / current picture is in Simple or Main Profile (No need to distinguish Simple and Main Profile) 1h / / current picture is in Advanced Profile
	10:6	**Reserved** Project: All Format: MBZ
	5	**Backward Prediction Present Flag** Note : a B picture that only uses forward prediction may have this flag set to 1 as well. Driver may still need to provide a valid reference picture index. This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicBackwardPrediction in DXVA2 VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode.
	4	**Intra Picture Flag** This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicIntra in DXVA2 VC1 spec. The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode.

MFD_VC1_SHORT_PIC_STATE

		Value / **Name** / **Description** 0h — entire picture can have a mixture of intra and inter MB type or just inter MB type. 1h — entire picture is coded in intra MB type
	3	**SecondField** This flag is set for the second field in field pictures.This field is used in both VLD and IT modes.
	2	**Reserved** Project: All Format: MBZ
	1:0	**Picture Structure** This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicStructure in DXVA2 VC1 spec. The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in DXVA2 VC1 VLD and IT mode. **Value** / **Name** / **Description** 01b — top field (bit 0) 10b — bottom field (bit 1) 11b — frame (both fields are present) 00b — illegal
3	31	**Reserved** Project: All Format: MBZ
	30	**Overlap Smoothing Enable Flag** This field is the decoded syntax element OVERLAP in bitstreamIndicates if Overlap smoothing is ON at the picture levelThis field is used in both VLD and IT modes **Value** / **Name** / **Description** / **Project** 0h — Disable — to disable overlap smoothing filter — All 1h — Enable — to enable overlap smoothing filter — All
	29	**Range Reduction Scale** Project: All Access: None This field specifies whether the reference picture pixel values should be scaled up or scaled down on-the-fly, if RangeReduction is Enabled.NOTE: This bit is derived by driver for Main Profile only. Ignored in Simple and Advanced Profiles. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture. RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1. For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

MFD_VC1_SHORT_PIC_STATE

Value	Name	Description	Project
0h	Disable **[Default]**	Scale down reference picture by factor of 2	All
1h	Enable	Scale up reference picture by factor of 2	All

28 **Range Reduction Enable**

This field specifies whether on-the-fly pixel value range reduction should be performed for the preceding (or forward) reference picture. Along with RangeReductionScale to specify whether scale up or down should be performed. It is not the same value as RANGEREDFRM Syntax Element (DXVA_PictureParameters bPicDeblocked bit 5) in the Picture Header.This field is for Main Profile only. Simple Profile is always disable, and not applicable to Advanced Profile. This field is used in both VLD and IT modes.This is derived by driver from the history of RANGERED and RANGEREDFRM syntax elements (i.e. of forward/preceding reference picture) and those of the current picture.RANGERED is the same as (bPicOverflowBlocks » 3) & 1. RANGEREDFRM is the same as (bPicDeblocked » 5) & 1.For the current picture is a B picture, this field represents the state of the forward/preceding reference picture onlyDriver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description	Project
0h	Disable **[Default]**	Range reduction is not performed	All
1h	Enable	Range reduction is performed	All

27:24 **Reserved**

23:22 **Progressive Pic Type**

This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicExtrapolation in DXVA2 VC1 spec.The Picture Structure and Progressive Pic Type are used to derive the picture structure as specified in FCM, in DXVA2 VC1 VLD and IT mode.

Value	Name	Description	Project
0		progressive only picture	All
1		progressive only picture	All
2		interlace picture (frame-interlace or field-interlace)	
3		illegal	

21 **Reserved**

20:16 **P-Pic Ref Distance**

Project:	All
Access:	None

This element defines the number of frames between the current frame and the reference frame. It is the same as the REFDIST SE in VC1 interlaced field picture header. It is present if the entry-level flag REFDIST_FLAG == 1, and if the picture type is not one of the following types: B/B, B/BI, BI/B, BI/BI. If the entry level flag REFDIST_FLAG == 0, REFDIST shall be set to the default value of 0.This field is used in DXVA2 VC1 VLD mode only, not used in IT and intel VC1 VLD Long Format modes.

Value	Name
0-16	unsigned integer
0h	**[Default]**

MFD_VC1_SHORT_PIC_STATE

15:14 QUANTIZER

Value	Name	Description
00b		implicit quantizer at frame leve
01b		explicit quantizer at frame level, and use PQUANTIZER SE to specify uniform or non-uniform
10b		explicit quantizer, and non-uniform quantizer for all frames
11b		explicit quantizer, and uniform quantizer for all frames

13 MULTIRES Present Flag (for Simple/Main Profile only)

Value	Name	Description
0h		RESPIC Parameter is present in the picture header
1h		RESPIC Parameter is present in the picture header

12 SYNCMARKER Present Flag (for Simple/Main Profile only)

Value	Name	Description
0		Bitstream for Simple and Main Profile has no sync marker
1		Bitstream for Simple and Main Profile may have sync marker(s)

11 RANGERED Present Flag (for Simple/Main Profile only)

It is needed for Picture Header Parsing.Driver is responsible to keep RangeReductionScale, RangeReduction Enable and RANGERED Present Flag of current picture coherent.

Value	Name	Description
0		Range Reduction Parameter (RANGEREDFRM) is not present in the picture header
1		Range Reduction Parameter (RANGEREDFRM) is present in the picture header.

10:8 MAXBFRAMES

Number of consecutive B Frames.

7 PANSCAN Present Flag

Value	Name	Description
0		Pan Scan Parameters are not present in the picture header
1		Pan Scan Parameters are present in the picture header

6 REFDIST_FLAG

For header parsing REFDIST.This is used in DXVA2 VC1 VLD mode only, not used in IT and intel VC1 VLD modes.

5 LOOPFILTER Enable Flag

This filed is the decoded syntax element LOOPFILTER in bitstream. It indicates if In-loop Deblocking is ON according to picture level bitstream syntax control. This bit affects BSD unit and also the loop filter unit.When this bit is set to 1, PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command must also be set to 1. In this case, in-loop deblocking operation follows the VC1 standard - deblocking doesn't cross slice boundary.When this bit is set to 0, but PostDeblockOutEnable field in MFX_PIPE_MODE_SELECT command is set to 1. It indicates the loop filter unit is used for out-of-loop deblocking. In this case, deblocking operation does cross slice boundary.This field is used in VLD mode only, not in IT mode.

MFD_VC1_SHORT_PIC_STATE

Value	Name	Description
0		In-Loop-Deblocking-Filter is disabled
1		In-Loop-Deblocking-Filter is enabled

4 **FastUVMCFlag (Fast UV Motion Compensation Flag)**

This field specifies whether the motion vectors for UV is rounded to half or full pel position. It is identical to the variable FASTUVMC in VC1 standard.This field is used in both VLD and IT modes.It is derived from FASTUVMC = (bPicSpatialResid8 » 4) & 1 in both VLD and IT modes, and should have the same value as Motion Vector Mode LSBit.

Value	Name	Description	Project
0h		no rounding	All
1h		quarter-pel offsets to half/full pel positions	All

3 **EXTENDED_MV Present Flag**

BitFieldDesc

Value	Name	Description	Project
0h		Extended_MV is not present in the picture header	All
1h		Extended_MV is present in the picture header	All

2:1 **DQUANT**

Project:	All
Access:	None
Format:	U2

Use for Picture Header Parsing of VOPDUANT elements

Value	Name	Description
0h	**[Default]**	
00b		no VOPDQUANT elements; Quantizer cannot vary in frame, same quantization step size PQUANT is used for all MBs in the frame
01b		refer to VC1 Spec. for all the MB position dependent quantizer selection
10b		The macroblocks located on the picture edge boundary shall be quantized with ALTPQUANT while the rest of the macroblocks shall be quantized with PQUANT.
11b	Reserved	

0 **VSTRANSFORM flag**

Value	Name	Description
0h	Disable	variable-sized transform coding is not enabled
1h	Enable	variable-sized transform coding is enabled

4 31:29 **Reserved**

Format:	MBZ (for possible future change to BFraction Enumeration)

28:24 **BFraction Enumeration**

MFD_VC1_SHORT_PIC_STATE

		This field is the scale factor for computing Direct-mode motion vectors. It is derived from the variable BFRACTION in the VC1 standard, section 8.4.5.4.There are only 21 valid values corresponding to the 21 encodings of BFRACTION as shown in the table here. Other values are reserved. The VLD decoded value of BFRACTION (from the picture header) is mapped into an enum value from 0 to 20. MSB of this field can be used to determine if BFRACTION is greater than or equal to 1/2, which is used to determine Motion Prediction Type for B pictures. Effectively, condition "BFRACTION >= 1/2" is equivalent to condition "ScaleFactor >= 128". This field is only valid for B pictures. This field is used only in DXVA2 VC1 VLD mode, it is not used in Intel VC1 VLD Long Format mode and IT mode.
	23	**Reserved** Project: All Format: MBZ Advanced Profile only; RANGE_MAPY_FLAG Range Mapping not supported
	22:20	**Reserved** Project: All Format: MBZ Advanced Profile only; RANGE_MAPY Range Mapping not supported
	19	**Reserved** Project: All Format: MBZ Advanced Profile only; RANGE_MAPUV_FLAG Range Mapping not supported
	18:16	**Reserved** Project: All Format: MBZ Advanced Profile only; RANGE_MAPUV Range Mapping not supported
	15:9	**Reserved** Project: All Format: MBZ
	8	**4MV Allowed Flag**
	7	**POSTPROC Flag**
	6	**PULLDOWN**
	5	**INTERLACE**
	4	**TFCNTRFLAG**
	3	**FINTERFLAG**
	2	**REFPIC Flag** For a BI picture, REFPIC flag must set to 0For I and P picture, REFPIC flag must set to 0.For a B picture, REFPIC flag must set to 0, except for a B-field in interlaced field mode which can be 0 or 1 (e.g. the top B field can be used as a reference for decoding its corresponding bottom B-field in a field pair).In VLD mode, this flag cannot be used as an optimization signaling for an I or P picture that is not used as a reference picture.This field is used in both DXVA2 VC1 VLD mode and IT mode. It is the same parameter as bPicDeblockConfined[bit2] in DXVA2 VC1 spec.The Intra Picture Flag, Backward Prediction Present Flag and RefPicFlag are used to derive the picture type, as specified in PTYPE for a frame, and in FPTYPE for a field, in DXVA2 VC1 VLD and IT mode.

MFD_VC1_SHORT_PIC_STATE

		Value / **Name** / **Description** 0h / / the current picture after decoded, will never used as a reference picture 1h / / the current picture after decoded, will be used as a reference picture later
	1	**PSF**
	0	**EXTENDED_DMV Present Flag** **Value** / **Name** / **Description** 0h / **[Default]** / Extended_DMV is not present in the picture header 1h / / Extended_DMV is present in the picture header

MFD_VP8_BSD_OBJECT

MFD_VP8_BSD_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MFD_VP8_BSD_OBJECT command is the only primitive command for the VP8 Decoding Pipeline. The Partitions of the bitstream is loaded as indirect data object. Before issuing a MFD_VP8_BSD_OBJECT command, all VP8 frame level states of the MFD Engine need to be valid. Therefore the commands used to set these states need to have been issued prior to the issue of a MFD_VP8_BSD_OBJECT command. Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFD_VP8_BSD_OBJECT Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8_DEC Format: OpCode
	23:21	**subOpcodeA** Default Value: 1h Format: OpCode
	20:16	**subOpcodeB** Default Value: 8h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 14h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:21	**Reserved** Format: MBZ
	20:16	**Partition0 CPBAC Entropy Count** Pass the Partition0 CPBAC State to HW. Max value is 24.
	15:8	**Partition0 CPBAC Entropy Range** Pass the Partition0 CPBAC State to HW.

MFD_VP8_BSD_OBJECT

	7:6	**Reserved** Format: MBZ
	5:4	**Coded Num of Coeff Token Partitions** Num of Partitions = 2^CodedNumCoeffTokenPariitons. 0 = 1 Partition only 1 = 2 Partitions 2 = 4 Partitions 3 = 8 Partitions are present in the bitstream.
	3	**Reserved** Format: MBZ
	2:0	**Partition0 FirstMBBitOffset from Frame Header** Allow HW to jump to the location in the bitstream where per MB information starts in the Partition0.
2	31:24	**Partition0 CPBAC Entropy Value** Pass the Partition0 CPBAC State to HW.
	23:0	**Reserved** Format: MBZ
3	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition0 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
4	31:0	**Indirect Partition0 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
5	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition1 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
6	31:0	**Indirect Partition1 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
7	31:24	**Reserved**

MFD_VP8_BSD_OBJECT

	23:0	**Indirect Partition2 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
8	31:0	**Indirect Partition2 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
9	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition3 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
10	31:0	**Indirect Partition3 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
11	31:24	**Reserved**
	23:0	**Indirect Partition4 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
12	31:0	**Indirect Partition4 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
13	31:24	**Reserved**
	23:0	**Indirect Partition5 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.

MFD_VP8_BSD_OBJECT		
14	31:0	**Indirect Partition5 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
15	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition6 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
16	31:0	**Indirect Partition6 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
17	31:24	**Reserved** Format: MBZ
	23:0	**Indirect Partition7 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
18	31:0	**Indirect Partition7 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.
19	31:24	**Reserved**
	23:0	**Indirect Partition8 Data Length** This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Partition Start Offset field is ignored. The Partition is byte aligned in both ends. It is the length in bytes of the bitstream data for the current partition. It includes the first byte of the first macroblock and the last byte of the last macroblock in the partition.
20	31:0	**Indirect Partition8 Data Start Offset** This field specifies the Graphics Memory starting address of the data to be fetched into BSD Unit for processing. This offset is relative to the MFD Indirect Object Base Address. Hardware ignores this field if indirect data is not present. It is a byte-aligned address for the VP8 bitstream data in each partition.

MFD_VP8_BSD_OBJECT

21 — 31 — Concealment Method

This field specifies the method used for concealment when error is detected.

Value	Name	Description
0	Intra 16x16 Prediction	A copy from the current picture is performed using Intra 16x16 Prediction method.
1	Inter P Copy	A copy from collocated macroblock location is performed from the concealment reference indicated by the ConCeal_Pic_Id field.

30:18 — Reserved

Format:	MBZ

17:16 — Conceal_Pic_Id (Concealment Picture ID)

Exists If:	[Concealment Method] == 1

This field identifies the picture in the reference list to be used for concealment. This field is only valid if Concealment Method is Inter P Copy. 00 - Last Decoded Picture 01 - Golden Reference Picture 02 - Alternate Reference Picture 03 - User provided Reference Picture

15 — Reserved

Format:	MBZ

14 — BSDPrematureComplete Error Handling

It occurs in situation where the decode is completed but there are still data in the bitstream.

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)

13 — Reserved

Format:	MBZ

12 — MPR Error (MV out of range) Handling

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)

11 — Reserved

10 — Entropy Error Handling

Value	Name
0	Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling
1	Set the interrupt to the driver (provide MMIO registers for MB address R/W)

9 — Reserved

Format:	MBZ

MFD_VP8_BSD_OBJECT

	8	**MB Header Error Handling** Value / Name: 0 — Ignore the error and continue (masked the interrupt), assume the hardware automatically performs the error handling. 1 — Set the interrupt to the driver (provide MMIO registers for MB address R/W)
	7:0	**Reserved** Format: MBZ

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_DIRECTMODE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a picture level command and is issued once per picture. All DMV buffers are treated as standard media surfaces, in which the lower 6 bits are used for conveying surface states.Current Pic POC number is assumed to be available in POCList[32 and 33] of the MFX_AVC_DIRECTMODE_STATE Command.This command is only valid in the AVC decoding in VLD and IT modes, and AVC encoder mode. The same command supports both Long and Short DXVA2 AVC Interface. The DMV buffers are not required to be programmed for encoder mode.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_SINGLE_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h Format: OpCode
	20:16	**SubOpcodeB** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0045h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:6	**Direct MV Buffer Base Address for Picture 0 (In Frame)** Format: GraphicsAddress[31:6] **Note:**This filed is changed to one per frame (both top and bottom field share the same Direct MV Buffer Base Address). This field provides the base address of the DMV write buffer to store motion vectors

MFX_AVC_DIRECTMODE_STATE

		decoded in the current picture (top field), which may be used later as a collocated motion information read buffer of the associated reference picture in decoding subsequent B-pictures that have MB coded in direct mode. It is a private buffer used by the MPR hardware only. Itscontent is not accessed by software. This buffer must be 64-byte cacheline aligned. The write buffer size is 557,056 bytes for 1 frame. Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution) It is only valid if the current picture is a progressive frame, MbAff frame, or a top field. There are a total of 32 reference picture (previously decoded) Direct MV Buffers (0 to 31, not including the DMV write buffer 32 and 33 of the current picture) to read in the corresponding collocated DMV and motion information. For reference picture, these 32 DMV read Buffers can be indexed by the frame_store_ID[4:0], which is obtained from RefPicList L0/L1[RefPicIdx]. frame_Store_IDbit[0] (indicator for Top/Bottiom Field). For writing out motion information during the decoding of the current picture, all 34 DMV buffers can be addressed by [img_dec_fs_idc[4:0]«1 + img_structure[1]].
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**Direct MV Buffer Base Address for Picture 0 - Read/Write [47:32]** Project: CHV, BSW Description: This field is for the upper range of AACS Bit Vector Surface Starting Byte Address. — Project: Description: This field is used for 48-bit addressing. — Project: CHV, BSW
3..32	63:48	**Reserved** Format: MBZ
	47:32	**Direct MV Buffer Base Address for Reference Frame 1 to 15 (In Frame) High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Pre-Deblocking Destination Address. This field is ignored if PreDeblockOutEnable is set to 0 (disable). This field is used for 48-bit addressing.
	31:6	**Direct MV Buffer Base Address for Reference Frame 1 to 15 (In Frame)** Format: GraphicsAddress[31:6]

MFX_AVC_DIRECTMODE_STATE

		Note: This field is changed to one per frame (both top and bottom field shared the same Direct MV Buffer Base Address) This field provides the base address of the DMV buffer for reference frame 2 to 31. They are needed if the current B-Picture has MBs coded in direct mode. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned. There are a total of 32 possible Direct MV Read Buffers (not including the current write buffer of the current picture) to read in the corresponding DMV. Each read buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). The adjacent DMV buffers are paired ([2 and 3], [4 and 5], [N and N+1], ..[30 and 31]).
	5:0	**Reserved** Format: MBZ Reserved for 64-bit address extension.
33	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Direct MV Buffer Base Address for Reference Frame - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Direct MV Buffer for Reference Picture 0 to 15**

MFX_AVC_DIRECTMODE_STATE

		Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** 00b: Use Cacheability Controls from page table 01b: Uncacheable (UC) - non-cacheable 10b: Writethrough (WT) 11b: Writeback (WB)
	4:3	**Target Cache (TC) for Direct MV Buffer for Reference Picture 0 to 15** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching **Value** / **Name** 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Direct MV Buffer for Reference Picture 0 to 15** Project: CHV, BSW This field allows the selection of AGE parameter for a given surface in LLC or eLLC. **Value** / **Name** 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
34	31:6	**Direct MV Buffer Base Address for Write (Write-Only Buffer)(in frame)** Format: GraphicsAddress[31:6] This field provides the base address of the DMV write-only buffer for the current decoding frame/field. It is a private buffer used by the MPR hardware only. Its content is not accessed by software. All these buffers must be 64-byte cacheline aligned, i.e. the same as the above DMV read/write buffers. These 2 buffers can only be addressed by [img_dec_fs_idc[4:0]«1 + img_structure[1]] for the current picture being decoded. Each write buffer size is 557,056 bytes for 1 frame (the selected colPic). Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution). DMV write buffer 32 is valid only if the current picture is a progressive frame, MbAff frame, or a top field. DMV write buffer 33 is valid only if the current picture is a bottom field.

MFX_AVC_DIRECTMODE_STATE

	5:0	**Reserved** Project: CHV, BSW Format: MBZ
35	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**Direct MV Buffer Base Address for Write (Write-Only Buffer)(in frame) High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Direct MV Buffer Base Address. This field is ignored if PreDeblockOutEnable is set to 0 (disable). This field is used for 48-bit addressing.
36	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved**
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Direct MV Buffer Base Address for Write - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Direct MV Buffer for Write** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name**

MFX_AVC_DIRECTMODE_STATE

	00b	Use Cacheability Controls from page table
	01b	Uncacheable (UC) - non-cacheable
	10b	Writethrough (WT)
	11b	Writeback (WB)

4:3 Target Cache (TC) for Direct MV Buffer for Write

Project:	CHV, BSW

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC
00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)*
01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*).
10b: LLC/eLLC Allowed.
11b: L3, LLC, eLLC Allowed.
Errata CHV:A-E (FIXED BY:G0 Stepping):
For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching.
For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops.
Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

1:0 Age for QUADLRU (AGE) for Direct MV Buffer for Write

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("0,1,2") it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits

MFX_AVC_DIRECTMODE_STATE			
		00b	Poor chance of generating hits
37..70	31:0	**POC List, POCList[34][31:0]** Each POC value is a signed 32-bit number. One-to-one correspondence with the 34 Direct MV Buffer Address for Reference and Currrent Frames/Fields There are 34 POC entries in the list. For reference picture, only the lower 32 POC [0-31] entries can be used, and POCList[ ] is indexed by the frame_store_ID[4:0], which is obtained from RefPicList L0/L1[RefPicIdx]. frame_Store_IDbit[0] (indicator for Top/Bottiom Field). For current picture, all 34 POC entries [0-33] can be addressed by POCList[img_dec_fs_idc[4:0]«1 + img_structure[1]]. For frame-only mode, every other entry is skipped. For MBAFF and field-only picture, each entry is a field POC, and every two entries are paired.	

MFX_AVC_IMG_STATE

MFX_AVC_IMG_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands. This command supports both Long and Short VLD and IT DXVA2 AVC Decoding Interface.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_IMG_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 1h AVC_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0Ch Excludes DWord (0,1) Format: =n 00Eh, used for normal decode and encode mode000h, a special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware.
1	31:16	**Reserved** Format: MBZ
	15:0	**Frame Size** Format: U16-1 in MB unit

MFX_AVC_IMG_STATE

		The value for FrameSizeInMBs must match the product of FrameWidthInMBs and FrameHeightInMBs.Max. Screen resolution is therefore limited to 256 x 256 in MB unit. It is enough to cover all the Profile-Level specified in the current HD-DVD specification. E.g., for 1920x1080, FrameSizeInMBs[15:0] = 8160 (1920/16 * 1088/16; rounded up 1080). This parameter is specified for Intel interface only, not present in the DXVA. **Value** / **Name** / **Description** [0,16383] / / representing Number of MBs [1,16384]
2	31:24	**Reserved** Format: MBZ (bit[31:24] must be zero to match the DXVA 16-bit definition for FrameHeightInMBsMinus1)
	23:16	**Frame Height** Format: U8-1 in MB unit It is set to the value of (FrameHeightInMBsMinus1+ 1). Since the max value for FrameHeightInMBs is 255, the max allowed value for FrameHeightInMBsMinus1 is only 254. The min value for FrameHeightInMBs is 1.Although the max. value that can be specified for FrameHeightInMBs is 255 (in the current implementation), FrameWidthInMBs * FrameHeightInMBs must not exceed the max value of FrameSizeInMBs[14:0].e.g. for 1920x1080, FrameHeightInMBs[7:0] is equal to 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).It is derived from FrameHeightInMbs = (2 - frame_mbs_only_flag) * PicHeightInMapUnits and PicHeightInMbs = FrameHeightInMbs / (1 + field_pic_flag) internally done. For MBAFF, PicHeightInMapUnits is in MB pair unit, so the bitstream sends only half frame height. **Value** / **Name** / **Description** [0,255] / / representing height [1,256]
	15:8	**Reserved** Format: MBZ (bit[15:8] must be zero to match the DXVA 16-bit definition for FrameWidthInMBsMinus1)
	7:0	**Frame Width** Format: U8-1 in MB unit It is set to the value of (FrameWidthInMBsMinus1+ 1). Since the max value for FrameWidthInMBs is 255, the max allowed value for FrameWidthInMBsMinus1 is only 254. The min value for FrameWidthInMBs is 1.Although the max. value that can be specified for FrameWidthInMBs is 255 (in the current implementation), FrameWidthInMBs * FrameWidthInMBs must not exceed the max value of FrameSizeInMBs[14:0].e.g. for 1920x1080, FrameHeightInMBs[7:0] is equal to 68 (1080 divided by 16, and rounded up, i.e. effectively specified as 1088 instead).It is derived from FrameWidthInMbs = (2 - frame_mbs_only_flag) * PicWidthInMapUnits and PicWidthInMbs = FrameWidthInMbs / (1 + field_pic_flag) internally done. For MBAFF, PicWidthInMapUnits is in MB pair unit, so the bitstream sends only half frame width.

MFX_AVC_IMG_STATE

		Value / **Name** / **Description**: [0,255] / / representing width [1,256]
3	31:29	**Reserved** Format: MBZ (bit[31:29] must be zero to match the DXVA2 8-bit definition for InitQpChroma[1])
	28:24	**Second Chroma QP Offset** Signed integer value. It should be in the range of -12 to +12 (according to AVC spec).It specifies the offset for determining QP Cr from QP Y. It is set to the upper 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS.Chroma_qp_offset [4:0] - chroma_qp_offset_bits (from the current active PPS)Chroma_qp_offset [9:5] - second_chroma_qp_offset_bits
	23:21	**Reserved** Format: MBZ (bit[23:21] must be zero to match the DXVA2 8-bit definition for InitQpChroma[1])
	20:16	**First Chroma QP Offset** Signed integer value. It should be in the range of -12 to +12 (according to AVC spec).It specifies the offset for determining QP Cb from QP Y. It is set to the lower 5 bits of the value of the syntax element (Chroma_qp_offset[9:0]) read from the current active PPS.Chroma_qp_offset [4:0] - chroma_qp_offset_bits (from the current active PPS)Chroma_qp_offset [9:5] - second_chroma_qp_offset_bits
	15:14	**Reserved** Format: MBZ
	13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**Weighted_Pred_Flag** Format: Enable (This field is defined differently from Gen6, Gen7 follows strictly DXVA2 AVC interface.) **Value** / **Name** / **Description**: 0 / Disable **[Default]** / specifies that weighted prediction is not used for P and SP slices 1 / Enable / specifies that weighted prediction is used for P and SP slices **Programming Notes** This field must set to '0' for B and I pictures.
	11:10	**Weighted_BiPred_Idc** **Value** / **Name** / **Description**

MFX_AVC_IMG_STATE

		0	DEFAULT **[Default]**	Specifies that the default weighted prediction is used for B slices
		1	EXPLICIT	Specifies that explicit weighted prediction is used for B slices
		2	IMPLICIT	Specifies that implicit weighted prediction is used for B slices.
		3	Reserved	Illegal value

Programming Notes
This field must set to 0 for P and I pictures.

	9:8	**ImgStruct - Image Structure, img_structure[1:0]** The current encoding picture structure can only takes on 3 possible values

Value	Name
00b	Frame Picture
01b	Top Field Picture
11b	Bottom Field Picture
10b	Invalid, not allowed.

Programming Notes
img_structure[0] can be used as a flag to distinguish between frame and field structure. It must be consistent with the field_pic_flag setting in the Slice Header.This parameter is specified for Intel interface only, not present in the DXVA as a separate state (instead the img_structure[1] is embedded inside the DXVA picture definition).

	7:0	**Reserved**

Format:	MBZ

4	31:16	**MinFrameWSize**

Default Value:	0h
Format:	U16

Minimum Frame Size [15:0] (in Word, 16-bit)(Encoder Only) Mininum Frame Size is specified to compensate for intel Rate Control Currently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Mininum Frame Size, is always less than maximum frame size **FrameBitRateMax (DWORD 10 bits** 29:16).This field is reserved in Decode mode.
The programmable range 0...2^18-1
When MinFrameWSizeUnits is 00.
Programmable range is 0...2^20-1 when MinFrameWSizeUnits is 01.
Programmable range is 0...2^26-1 when MinFrameWSizeUnits is 10.
Programmable range is 0...2^32-1 when MinFrameWSizeUnits is 11.

MFX_AVC_IMG_STATE

15 **MbStatEnabled**

Format:	Enable

Enable reading in MB status buffer (a.k.a. encoding stream-out buffer) Note: For multi-pass encoder, all passes except the first one need to set this value to 1. By setting the first pass to 0, it does save some memory bandwidth.
In VDenc mode this must be set to zero as no MB level rate control is used.

Value	Name	Description
0	Disable	Disable Reading of Macroblock Status Buffer
1	Enable	Enable Reading of Macroblock Status Buffer

14 **LoadSlicePointerFlag**

Format:	Enable

LoadBitStreamPointerPerSlice (Encoder-only)To support multiple slice picture and additional header/data insertion before and after an encoded slice.When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream.When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0	Disable	Load BitStream Pointer only once for the first slice of a frame
1	Enable	Load/reload BitStream Pointer only once for the each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field

13 **Reserved**

12 **MvUnpackedFlag**

MVUnPackedEnable (Encoder Only)This field is reserved in Decode mode.

Value	Name	Description
0	PACKED	use packed MV format (compliant to DXVA)
1	UNPACKED	use unpacked 8MV/32MV format only

11:10 **ChromaFormatIdc**

Chroma Format IDC, ChromaFormatIdc[1:0]It specifies the sampling of chroma component (Cb, Cr) in the current picture as follows :

Value	Name	Description
00b	monochrome picture	Desc
01b	4:2:0 picture	Desc
10b	4:2:2 picture (not supported)	
11b	4:4:4 picture (not supported)	

Programming Notes
It is set to the value of the syntax element read from the current active SPS.The

MFX_AVC_IMG_STATE

corresponding Monochrome Flag (monochrome_flag) can be derived from this field.

9 **Reserved**

8 **MbMvFormatFlag**

Use MB level MvFormat flag (Encoder Only)

Value	Name	Description
0	IGNORE	HW PAK ignore MvFormat in the MB data. When bit 12 == 0, all MBs use packed MV formatWhen bit 12 == 1, each MB data must use unpacked MV format, 8MV when there is no minor MV involved, and 32MV if there are some minor MVs.
1	FOLLOW	HW PAK will follow MvFormat value set within each MB data.

Programming Notes

They must take one of the two values: the 8MV unpacked format (MvFormat =101b), and the 32MV unpacked format (MvFormat =110b).This bit can be set only when MvUnpackedFlag (bit 12 of this register) is set otherwise system could hang.

7 **EntropyCodingFlag**

Entropy Coding Flag, entropy_coding_flag

Value	Name	Description
0	CAVLC bit-serial encoding mode	Desc
1	CABAC bit-serial encoding mode.	Desc

Programming Notes

It specifies one of the two possible bit stream encoding modes in the AVC. It is set to the value of the syntax element read from the current active PPS.

6 **ImgDisposableFlag**

Current Img Disposable Flag or Non-Reference Picture Flag

Value	Name	Description
0	REFERENCE	the current decoding picture may be used as a reference picture for others
1	DISPOSABLE	the current decoding picture is not used as a reference picture (e.g. a B-picture cannot be a reference picture for any subsequent decoding)

Programming Notes

It is derived from ImgDisposableFlag = (nal_ref_idc == 0). nal_ref_idc is a syntax element from a NAL unit. When this flag is set, no reference picture and DMV are written out.This field is only valid for VLD decoding mode.

5 **ConstrainedIPredFlag**

Constrained Intra Prediction Flag, constrained_ipred_flagIt is set to the value of the syntax element in the current active PPS.

MFX_AVC_IMG_STATE

Value	Name	Description
0	INTRA_AND_INTER	allows both intra and inter neighboring MB to be used in the intra-prediction encoding of the current MB.
1	INTRA_ONLY	allows only to use neighboring Intra MBs in the intra-prediction encoding of the current MB. If the neighbor is an inter MB, it is considered as not available.

4 **Direct8x8InfFlag**

Direct 8x8 Inference Flag, direct_8x8_inference_flagIt is set to the value of the syntax element in the current active SPS.It specifies the derivation process for luma motion vectors in the Direct MV coding modes (B_Skip, B_Direct_16x16 and B_Direct_8x8). When frame_mbs_only_flag is equal to 0, direct_8x8_inference_flag shall be equal to 1.It must be consistent with the frame_mbs_only_flag and transform_8x8_mode_flag settings.

Value	Name	Description
0	SUBBLOCK	allows subpartitioning to go below 8x8 block size (i.e. 4x4, 8x4 or 4x8)
1	BLOCK	allows processing only at 8x8 block size. MB Info is stored for 8x8 block size.

3 **Transform8x8Flag**

8x8 IDCT Transform Mode Flag, trans8x8_mode_flagSpecifies 8x8 IDCT transform may be used in this pictureIt is set to the value of the syntax element in the current active PPS.

Value	Name	Description
0	4x4	no 8x8 IDCT Transform, only 4x4 IDCT transform blocks are present
1	8x8	8x8 Transform is allowed

2 **FrameMbOnlyFlag**

Frame MB only flag, frame_mbs_only_flagIt is set to the value of the syntax element in the current active SPS.

Value	Name	Description
0	FALSE	not true ; effectively enables the possibility of MBAFF mode.
1	TRUE	true, only frame MBs can occur in this sequence, hence disallows the MBAFF mode and field picture.

1 **MbaffFlameFlag**

MBAFF mode is active, mbaff_frame_flag.It is derived from MbaffFrameFlag = (mb_adaptive_frame_field_flag && ! field_pic_flag). mb_adaptive_frame_field_flag is a syntax element in the current active SPS and field_pic_flag is a syntax element in the current Slice Header. They both are present only if frame_mbs_only_flag is 0. Although mbaff_frame_flag is a Slice Header parameter, its value is expected to be the same for all the slices of a picture.It must be consistent with the mb_adaptive_frame_field_flag, the field_pic_flag and the frame_mbs_only_flag settings.This bit is valid only when the img_structure[1:0] indicates the current picture is a frame.

MFX_AVC_IMG_STATE

Value	Name	Description
0	FALSE	not in MBAFF mode
1	TRUE	in MBAFF mode

FieldPicFlag

Field picture flag, field_pic_flag, specifies the current slice is a coded field or not.It is set to the same value as the syntax element in the Slice Header. It must be consistent with the img_structure[1:0] and the frame_mbs_only_flag settings.Although field_pic_flag is a Slice Header parameter, its value is expected to be the same for all the slices of a picture.

Value	Name	Description
0h	FRAME	a slice of a coded frame
1h	FIELD	a slice of a coded field

5 [ExistsIf]Encode Only

31 **Trellis Quantization Enabled (TQEnb)**

Format:	Enable

The TQ improves output video quality of AVC CABAC encoder by selecting quantized values for each non-zero coefficient so as to minimize the total R-D cost.This flag is only valid AVC CABAC mode. Otherwise, this flag should be disabled.

Value	Name	Description	Project
0h	Disable	Use Normal	
1h	Enable	Use Trellis quantization	CHV, BSW

30:28 **Trellis Quantization Rounding (TQR)**

This rounding scheme is only applied to the quantized coefficients ranging from 0 to 1 when TQEnb is set to 1 in AVC CABAC mode. One of the following values is added to quantized coefficients before truncating fractional part.

Value	Name	Description	Project
000b		Add 1/8	CHV, BSW
001b		Add 2/8	CHV, BSW
010b		Add 3/8	CHV, BSW
011b		Add 4/8 (rounding 0.5)	CHV, BSW
100b		Add 5/8	CHV, BSW
101b		Add 6/8	CHV, BSW
110b	Default	Add 7/8 (Default rounding 0.875)	CHV, BSW

27 **Trellis Quantization Chroma Disable (TQChromaDisable)**

This signal is used to disable chroma TQ. To enable TQ for both luma and chroma, TQEnb=1, TQChromaDisable=0. To enable TQ only for luma, TQEnb=1, TQChromaDisable=1.

Value	Name	Description	Project
0h		Enable Trellis Quantization chroma	CHV, BSW
1h	Default	Disable Trellis Quantization chroma	CHV, BSW

MFX_AVC_IMG_STATE

26:21 Reserved

Project:	CHV, BSW
Format:	MBZ

20:17 Reserved

Format:	MBZ

16 NonFirstPassFlag

This signals the current pass is not the first pass. It will imply designate HW behavior: e.g

Value	Name	Description
0h	Disable	Always use the MbQpY from initial PAK inline object for all passes of PAK
1h	Enable	Use MbQpY from stream-out buffer if MbRateCtrlFlag is set to 1

15:13 Reserved

Format:	MBZ

12 Reserved

Project:	CHV, BSW
Format:	MBZ

11:10 MinFrameWSizeUnits

This field is the Minimum Frame Size Units

Value	Name	Description
00b	compatibility mode	Minimum Frame Size is in old mode (words, 2bytes)
01b	16 byte	Minimum Frame Size is in 16bytes
10b	4Kb	Minimum Frame Size is in 4Kbytes
11b	16Kb	Minimum Frame Size is in 16Kbytes

9 MbRateCtrlFlag - MB level Rate Control Enabling Flag

MB Rate Control conformance mask

In VDenc mode, this field must be zero as frame level rate control is used.

Value	Name	Description
0h	Disable	Apply accumulative delta QP for consecutive passes on top of the macroblock QP values in inline data
1h	Enable	Apply RC QP delta to suggested QP values in Macroblock Status Buffer except the first pass.

Programming Notes
This field is ignored when MacroblockStatEnable is disabled or MB level Rate control flag for the current MB is disable in Macroblock Status Buffer.

8 Reserved

Format:	MBZ

MFX_AVC_IMG_STATE

7 — **Intra/InterMbIpcmFlag - ForceIPCMControlMask**

This field is to Force **IPCM** for Intra or Inter Macroblock size conformance mask.

Value	Name	Description	Project
0h	Disable	Do not change intra or Inter macroblocks even	CHV, BSW
1h	Enable	Change intra or Inter macroblocks MB_type to IPCM	CHV, BSW

Programming Notes
This field is ignored when MacroblockStatEnable is disabled or MB level Intra MB conformance flag for the current MB is disable in Macroblock Status Buffer.

6:4 — **Reserved**

Format:	MBZ

3 — **FrameSzUnderFlag - FrameBitRateMinReportMask**

This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin

Value	Name	Description
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1h	Enable	set bit0 and bit 1of MFC_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit rate Minimum limit.

2 — **FrameSzOverFlag - FrameBitRateMaxReportMask**

This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description
0	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1	Enable	Set bit0 and bit 1 of MFC_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit rate Maximum limit.

1 — **InterMbMaxBitFlag - InterMBMaxSizeReportMask**

This is a mask bit controlling if the condition of any inter MB in the frame exceeds InterMBMaxSize.

Value	Name	Description
0	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.
1	Enable	Set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Inter MB Conformance Max size limit.

0 — **IntraMbMaxBitFlag - IntraMBMaxSizeReportMask**

This is a mask bit controlling if the condition of any intra MB in the frame exceeds IntraMBMaxSize.

Value	Name	Description
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.

MFX_AVC_IMG_STATE

		1 Enable set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Intra MB Conformance Max size limit.
6 [ExistsIf]Encode Only	31:28	**Reserved**
	27:16	**InterMbMaxSz** Format: U12 This field, Inter MB Conformance Max size limit,indicates the allowed max bit count size for Inter MB
	15:12	**Reserved** Format: MBZ
	11:0	**IntraMbMaxSz** Exists If: //Intra Only Format: U12 This field, Intra MB Conformance Max size limit,indicates the allowed max bit count size for Intra MB All IPCM MBs should ignore this Max size limit.
7 [ExistsIf]Encode Only	31:1	**Reserved**
	0	**VSL Top MB Trans8x8flag** Project: CHV, BSW **Value** / **Name** / **Description** 0 Disable **[Default]** VSL will only fetch the current MB data. 1 Enable When this bit is set VSL will make extra fetch to memory to fetch the MB data for top MB.
8 [ExistsIf]Encode Only	31:24	**SliceDeltaQpMax[3]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for total bit-count above FrameBitRateMax - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta»3).
	23:16	**SliceDeltaQpMax[2]** Format: U8

MFX_AVC_IMG_STATE

		Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/8 and below 1/ 4 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and ¼ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»3), (FrameBitRateMax+ FrameBitRateMaxDelta»2).
	15:8	**SliceDeltaQpMax[1]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above1/ 4 and below 1/2 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between ¼ and ½ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»2), (FrameBitRateMax+ FrameBitRateMaxDelta»1).
	7:0	**SliceDeltaQpPMax[0]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta , i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»1), infinite).
9 [ExistsIf]Encode Only	31:24	**SliceDeltaQpMin[3]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for total bit-count below FrameBitRateMin - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin-FrameBitRateMinDelta»3), FrameBitRateMin).
	23:16	**SliceDeltaQpMin[2]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count below FrameBitRateMin - below 1/ 8

MFX_AVC_IMG_STATE

		and above 1/ 4This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»2), (FrameBitRateMin- FrameBitRateMinDelta»3)).
	15:8	**SliceDeltaQpMin[1]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice level delta QP for bit-count below FrameBitRateMin- below 1/4 and above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»1), (FrameBitRateMin- FrameBitRateMinDelta»2)).
	7:0	**SliceDeltaQpMin[0]** Format: S7 Range: [0:MAX_QP_DELTA] This field is the Slice Level Delta QP for bit-count below FrameBitRateMin - below 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta , i.e., in the range of [0, (FrameBitRateMin- FrameBitRateMinDelta»1).
10 [ExistsIf]Encode Only	31	**FrameBitrateMaxUnit** This field is the Frame Bitrate Maximum Limit Units. Value 0 / Name Byte / Description: FrameBitRateMax is in units of 32 Bytes when FrameBitrateMaxUnitMode is 1 and in units of 128 Bytes if FrameBitrateMaxUnitMode is 0 Value 1 / Name Kilo Byte / Description: FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0
	30	**FrameBitrateMaxUnitMode** This field is the Frame Bitrate Maximum Limit Units. Value 0h / Name compatibility mode / Description: FrameBitRateMaxUnit is in old mode (128b/16Kb) Value 1h / Name New mode / Description: FrameBitRateMaxUnit is in new mode (32byte/4Kb)
	29:16	**FrameBitRateMax** This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when

MFX_AVC_IMG_STATE

enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28 and 29 should be 0..

Value	Name	Description
0-512KB		The programmable range is 0-512KB when FrameBitrateMaxUnit is 0.
0-8190KB		The programmable range is 0-8190KB when FrameBitrateMaxUnit is 1.

15 **FrameBitrateMinUnit**

This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description
0	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMinUnitMode is 1 and in units of 128 Bytes if FrameBitrateMinUnitMode is 0
1	Kilo Byte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0

14 **FrameBitrateMinUnitMode**

This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description
0h	Compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)
1h	New mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)

13:0 **FrameBitRateMin**

RangeThe programmable range 0-512KB When FrameBitrateMinUnit is in 0.Programmable range is 0-8190 KB when FrameBitrateMinUnit is in 1.This field is the Frame Bitrate Minimum Limit ()This field along with FrameBitrateMinUnit determines minimum allowed bits in a Frame before Multi-Pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count is less than this value. When FrameBitrateMinUnitMode is 0 (compatibility mode) bits 0:11 should be used, bits 12 and 13 should be 0.

11 [ExistsIf]Encode Only

31 **Slice Stats Streamout Enable**

30:16 **FrameBitRateMaxDelta**

Format:	U15

This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28, 29 and 30 should be 0.

Value	Name	Description
0-1024KB		The Programmable range 0-1024KB when FrameBitRateMaxUnit is 0.
0-16380KB		The Programmable range is 0-16380KB when FrameBitRateMaxUnit is 1.
0h	**[Default]**	

MFX_AVC_IMG_STATE

	15	**Reserved** Format: MBZ
	14:0	**FrameBitRateMinDelta** Range: The programmable range 0-1024KB When FrameBitrateMinUnit is in 32Bytes.Programmable range is 0-16380KB when FrameBitrateMinUnit is in 4Kbytes. This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. When FrameBitrateMinUnitMode is 0(compatibility mode) bits 0:11 should be used, bits 12, 13 and 14 should be 0.Note: HW requires the following condition FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior.
12	31:21	**Reserved**
	20	**VMD Error Logic** Project: CHV, BSW Value / Name / Description: 0 — Disable **[Default]** 1 — Enable — Error Handling
	19	**Reserved**
	18	**VAD Error Logic** Project: CHV, BSW Value / Name / Description: 0 — Enable **[Default]** — Error reporting ON in case of premature Slice done 1 — Disable — CABAC Engine will auto decode the bitstream in case of premature slice done.
	17	**Reserved**
	16	**Reserved**
	15:0	**Reserved**
13	31:30	**Reserved**
	29	**Current Picture Has Performed MMCO5** Set to 1 if the current Pic has performed the memory_management_control_operation = = 5.
	28:24	**Number of Reference Frames** Format: U5 Range: Range 0 to MaxDpbSize (=16 for Level 4.1) Specifies the maximum number of reference frames (frames, field pairs, unpaired field) existed in the current DBP for decoding the current picture.

MFX_AVC_IMG_STATE

	23:22	**Reserved** Format: MBZ
	21:16	**Number of Active Reference Pictures from L1** Format: U6-1 Specifies the initial maximum reference index value minus 1 to access the L1 Reference List. It is extracted from PPS. It corresponds to the number of active reference pictures from L1 to decode the current picture. It can be modified by the slice header if num_ref_idx_active_override_flag is set. Only valid for B picture. Value: [0,31] — Name:
	15:14	**Reserved** Format: MBZ
	13:8	**Number of Active Reference Pictures from L0** Format: U6-1 Specifies the initial maximum reference index value minus 1 to access the L0 Reference List. It is extracted from PPS. It corresponds to the number of active reference pictures from L0 to decode the current picture. It can be modified by the slice header if num_ref_idx_active_override_flag is set. Valid for both P and B pictures. Value: [0,31] — Name:
	7:0	**Initial QP Value** Format: S7 Range: [-26,25] Initial QP value for a Slice, extracted from PPS. It may further get modified by slice_qp_delta in slice header and mb_qp_delta in MB header.
14 [ExistsIf] Short Format only	31:24	**Log2_max_pic_order_cnt_lsb_minus4** Exists If: //Short Format Only It is a SPS syntax element, used to determine how many bits in the bitstream are used to represent pic_order_cnt_lsb syntax element in the slice header.Unsigned
	23:16	**Log2_max_frame_num_minus4** Exists If: //Short Format Only It is a SPS syntax element, used to determine how many bits in the bitstream are used to represent frame_num syntax element in the slice header.Unsigned.
	15	**deblocking_filter_control_present_flag** Exists If: //Short Format Only It is a PPS syntax element, indicates if more deblocking filter control syntax elements are present in the slice header.

MFX_AVC_IMG_STATE

	14:12	**num_slice_groups_minus1** Exists If: //Short Format Only BitField It is a PPS syntax element.Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.Desc
	11	**redundant_pic_cnt_present_flag** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only, to read-in redundant_pic_cnt, if any, but is not used by H/W, i.e. no support for redundant slice processing.
	10:8	**slice_group_map_type** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.
	7:4	**Reserved**
	3:2	**Pic_order_cnt_type** Exists If: //Short Format Only It is a SPS syntax element.Use for Slice Header parsing only.
	1	**Delta_pic_order_always_zero_flag** Exists If: //Short Format Only It is a SPS syntax element.Use for Slice Header parsing only.
	0	**Pic_order_present_flag** Exists If: //Short Format Only It is a PPS syntax element.Use for Slice Header parsing only.
15 [ExistsIf] Short Format only	31:16	**Curr Pic Frame Num** Exists If: //Short Format Only Format: U16 Derived from Slice Header syntax element
	15:0	**Slice Group Change Rate** Exists If: //Short Format Only Format: U16-1 It is a PPS syntax element Use for Slice Header parsing only, to read in slice_group_change_cycle, if any, but is not used by H/W, i.e. no slice group support.

MFX_AVC_IMG_STATE

16

[ExistsIf]: Short Format only

31 **Inter View Order Disable**

Project:	CHV, BSW
Exists If:	//Short Format Only

It indicates how to append inter-view picture into initial sorted reference list. (due to ambiguity in the MVC Spec)

Value	Name	Description
0h	Default **[Default]**	View Order Ascending
1h	Disable	View ID Ascending

30:22 **Reserved**

Project:	CHV, BSW
Format:	MBZ

21:18 **Max View IDXL1**

Project:	CHV, BSW
Exists If:	//Short Format Only

It is a PPS syntax element corresponding to Anchor/Non-Anchor Reference ListL1 It indicates the maximum number of inter-view picture for Reference List L1

17:16 **Reserved**

Project:	CHV, BSW
Format:	MBZ

15:12 **Max View IDXL0**

Project:	CHV, BSW
Exists If:	//Short Format Only

Reference ListL0 It indicates the maximum number of inter-view picture for Reference List L0

11:10 **Reserved**

Project:	CHV, BSW
Format:	MBZ

9:0 **Current Frame View ID**

Project:	CHV, BSW
Exists If:	//Short Format Only

It indicates the View ID of the current decoding frame

MFX_AVC_REF_IDX_STATE

MFX_AVC_REF_IDX_STATE

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD mode); it is not need in decoder IT mode.
The inline data of this command is interpreted differently for encoder as for decoder. For decoder, it is interpreted as RefIdx List L0/L1 as in AVC spec., and it matches with the DXVA2 AVC API data structure for decoder in VLD mode : RefPicList[2][32] (L0:L1, 0:31 RefPic). But for encoder, it is interpreted as a Reference Index Mapping Table for L0 and L1 reference pictures. For packing the bits at the output of PAK, the syntax elements must follow the definition of RefIdxL0/L1 list according to the AVC spec. However, the decoder pipeline was designed to use a variation of that standard definition, as such a conversion (mapping) is needed to support the hardware design.
The Reference lists are needed in processing both P and B slice in AVC codec. For P-MB, only L0 list is used; for B-MB both L0 and L1 lists are needed. For a B-MB that is coded in L1-only Prediction, only L1 list is used.

Programming Notes

DXVA2 specifies that an application will create the RefPicList L0 and L1 and pass onto the driver. The content of each entry of RefPicList L0/L1[ ] is a 7-bit picture index. This picture index is the same as that of RefFrameList[ ] content. This picture index, however, is not defined the same as the frame store ID (0 to 16, 5-bits) we have implemented in H/W. Hence, driver is required to manage a table to convert between DXVA2 picture index and intel frame store ID. As such, the final RefPicList L0/L1[ ] that the driver passes onto the H/W is not the same as that defined in the DXVA2.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_REF_IDX_STATE Format: OpCode
	26:24	**Command Opcode** Default Value: 1h AVC Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h MFX_AVC_REF_IDX_STATE Format: OpCode

MFX_AVC_REF_IDX_STATE

	20:16	**SubOpcodeB** Default Value: 4h MFX_AVC_REF_IDX_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0008h Format: =n Excludes DWords 0,1
1	31:1	**Reserved** Format: MBZ
	0	**RefPicList Select** Num_ref_idx_l1_active is resulted from the specifications in both PPS and Slice Header for the current slice. However, since the full reference list L0 and/or L1 are always sent, only present flags are specified instead. This parameter is specified for Intel interface only, not present in the DXVA.

Value	Name	Description
0	RefPicList 0	The list that followed represents RefList L0 (Decoder VLD mode) or Ref Idx Mapping Table L0 (Encoder PAK mode)
1	RefPicList1	The list that followed represents RefList L1 (Decoder VLD mode) or Ref Idx Mapping Table L1 (Encoder PAK mode)

2..9	31:0	**Reference List Entry** This set of fields is always present whenever this command is issued. It always specifies the full 32 reference pictures in the selected list, regardless they are "existing picture" or not. If a picture is non-existing, the corresponding entry should be set to all ones. Each list entry is 1 byte. A 32-bit DW can hold 4 list entries in the following format • 31:24 entry X+3 (e.g. listY_3) • 23:16 entry X+2 (e.g. listY_2) • 15:8 entry X+1 (e.g. listY_1) • 7:0 entry X (e.g. listY_0) X is replaced by the paddr[2:0] * 4 ; paddr[5:0] with 0x20 and 0x27, and Y is replaced by 0 or 1. The byte definition for a reference picture : • Bit 7 : Non-Existing - indicates that frame store index that should have been at this entry did not exist and was replaced by an index 0 (a valid entry) for error concealment • Bit 6 : Long term bit - set this reference picture to be used as long term reference • Bit 5 : Field picture flag - indicates frame/field • Bit 4:0 : Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)

MFX_AVC_REF_IDX_STATE		
		This is the final Reference List L0 or L1 after any reordering specified in the Slice Header as well as modified by the driver, and its indices values are all translated to the intel specification. If the reference picture is a frame (Bit5 = 1), frame store ID is always an even number. This list is used in outputting MV information by the BSD unit in VLD mode. DMV access also reads and writes Mvlist0 using this frame store ID. If this set of fields is interpreted as Reference Index Mapping Table L0/L1, the same field alignment is followed, i.e. 4 mapping entries per DW. Each mapping entry is one byte in size, but only the least significant 5 bits [4:0] is relevant. Driver should zero all the upper bits [7:5] for each entry.

MFX_AVC_SLICE_STATE

MFX_AVC_SLICE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes).

Programming Notes

MFX_AVC_SLICE_STATE command is not issued for AVC DXVA2 Short Format Bitstream decode, instead MFD_AVC_SLICEADDR command is executed to retrieve the next slice MB Start Address X and Y by H/W itself.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_AVC_SLICE_STATE Format: OpCode
	26:24	**Command Opcode** Default Value: 1h AVC Format: OpCode
	23:21	**SubOpcodeA** Default Value: 0h MFX_AVC_SLICE_STATE Format: OpCode
	20:16	**Command SubOpcodeB** Default Value: 3h MFX_AVC_SLICE_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 8h DWORD_COUNT_n Format: =n Excludes DWords 0,1
1	31:4	**Reserved** Format: MBZ

MFX_AVC_SLICE_STATE

	3:0	**Slice Type** It is set to the value of the syntax element read from the Slice Header. **Value** / **Name** 0000b — P Slice 0001b — B Slice 0010b — I Slice 0011b-1111b — Reserved **Programming Notes** Bits[3:2] must be 0
2	31:30	**Reserved** Format: MBZ
	29:24	**Number of Reference Pictures in Inter-prediction List 1** Format: U6 This field is valid only for encoding a B Slice, for which it is expected to have at least one entry in the reference list L1; otherwise (if Slice Type is not a B Slice), this field must be set to 0. This field can be derived for a B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L1 = NumRefIdxActiveMinus1[1] + 1. **Value** / **Name** 0-32
	23:22	**Reserved** Format: MBZ
	21:16	**Number of Reference Pictures in Inter-prediction List 0** Format: U6 This field is valid for encoding a P or B Slice, for which it is expected to have at least one entry in the reference list L0; otherwise (if Slice Type is not a P or B Slice), this field must be set to 0. This field can be derived for a P or B Slice from the Slice Header syntax element NumRefIdxActiveMinus1 as, Num_Ref_Idx_L0 = NumRefIdxActiveMinus1[0] + 1. **Value** / **Name** 0-32
	15:11	**Reserved**
	10:8	**Log 2 Weight Denom Chroma** Format: U3 **Value** / **Name** 0-7
	7:3	**Reserved**

MFX_AVC_SLICE_STATE

	2:0	**Log 2 Weight Denom Luma** Format: U3 It is the base 2 logarithm of the denominator for all Luma weighting factors. It is set to the value of the syntax element read from the Slice Header Pred_Weight_Table(). **Value** / **Name** 0-7
3	31:30	**Weighted Prediction Indicator** This field indicates the Weighted Prediction mode for a P or B Slice. It is a combined field corresponding to the syntax element WeightedBiPredIdc or WeightedPredFlag read from the current active PPS. • If it is a B-Slice, these bits are interpreted as: 00b - Specifies the default weighted inter-prediction to be applied 01b - Specifies the explicit weighted inter-prediction to be applied 10b - Specifies the implicit weighted inter-prediction to be applied 11b - Reserved (not allowed) • If it is a P Slice, these bits are interpreted as: 00b - Disables weighted inter-prediction (Default weighted) 01b - Enables weighted inter-prediction (Explicit weighted) 10b - 11b - Reserved **Programming Notes** Only when in B Slice with Weighted_Pred_Idc = 1 (explicit weighted prediction), will there be a L1 and/or a L0 weight+offset tables being sent to the BSD unit through the Slice_State command. Only when in P Slice with Weighted_Pred_Idc = 1, will there be a L0 weight+offset table being sent to the BSD. If Weighted_Pred_Idc != 1 for B Slice or Weighted_Pred_Idc =0 for P Slice, no Slice_State command should be issued to send these tables. If still being issued, the data is read but ignored. DXVA specifies Weighted_Bipred and Weighted_Pred in frame-level state. However, these two flags are combined and specified in slice level for both P and B slice type.
	29	**Direct Prediction Type** Type of direct prediction used for B Slices. This field is valid only for Slice_Type = B Slice; otherwise, it must be set to 0. **Value** / **Name** 0 / Temporal 1 / Spatial
	28:27	**Disable Deblocking Filter Indicator** **Value** / **Name** / **Description** 00b / / FilterInternalEdgesFlag is set equal to 1 01b / / Disable all deblocking operation, no deblocking parameter syntax element is read; filterInternalEdgesFlag is set equal to 0 10b / / Macroblocks in different slices are considered not available;

MFX_AVC_SLICE_STATE

		filterInternalEdgesFlag is set equal to 1
11b	Reserved	Not defined in AVC

26 — Reserved

Format:	MBZ

25:24 — Cabac Init Idc[1:0]

Specifies the index for determining the initialization table used in the context variable initialization process.

Value	Name
0-2	

Programming Notes
Cabac initialization is also dependent on the field/frame picture type, Slice type, and the current SliceQP value.

23:22 — Reserved

Format:	MBZ

21:16 — Slice Quantization Parameter

Quantization Parameter for current slice. Derived from PPS and slice_delta_qp syntax element in Slice Header. It is needed for CABAC context initialization and deblocking filter control. And it is also used as the starting QP value in the very first MB of a slice. It is in the range of unsigned integer 0 to 51, for 8-bit pixel bit-depth.

15:12 — Reserved

Format:	MBZ

11:8 — Slice Beta Offset Div2

Format:	S3 2's Complement

Range: [-6, 6] Inclusive
Specifies the offset used in accessing the deblocking filter strength tables.

7:4 — Reserved

Format:	MBZ

3:0 — Slice Alpha C0 Offset Div2

Format:	S3 2's Complement

Range: [-6, 6] Inclusive
Specifies the offset used in accessing the deblocking filter strength tables.

4 / 31:24 — Slice Vertical Position

This field specifies the position in y-direction of the first macroblock in the Slice in unit of macroblocks. The fields (Slice_MB_Start_Hor_Pos, Slice_MB_Start_Vert_Pos) are valid in VLD (decoding) mode only. They are ignored by hardware in decoding IT mode and encoding mode (whereas the position is provided by the per-macroblock object command). Derived

MFX_AVC_SLICE_STATE

		Programming Notes Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	23:16	**Slice Horizontal Position** This field specifies the position in x-direction of the first macroblock in the Slice in unit of macroblocks. Derived **Programming Notes** Error Handling: Driver needs to check if FirstMbY starts at 0 on the first slice of frame. If not, driver needs to add a phantom slice with FirstMbX and FirstMbY set to 0.
	15	**Reserved** Format: MBZ
	14:0	**Slice Start Mb Num** Exists If: //Decoder Only The MB number (linear MB address in a picture) at the start of a Slice, it must match with the Slice Horizontal Position (Slice_MB_Start_Hor_Pos) and Vertical Position (Slice_MB_Start_Vert_Pos) in the picture. **Programming Notes** In creating the Phantom Slice for error concealment, this field should set to the total number of MB in the current picture + 1.
5	31:24	**Reserved** Format: MBZ
	23:16	**Next Slice Vertical Position** This field specifies the position in y-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of picture (since y-direction is zero-based numbering).
	15:8	**Reserved** Format: MBZ
	7:0	**Next Slice Horizontal Position** This field specifies the position in x-direction of the first macroblock in the next Slice in unit of macroblocks. This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to 0.
6 Encoder Only	31	**Rate Control Counter Enable** To enable the accumulation of bit allocation for rate control This field enables hardware Rate Control logic. The rest of the RC control fields are only valid when this field is set to 1. Otherwise, hardware ignores these fields. **Value** / **Name** 0 – Disable 1 – Enable

MFX_AVC_SLICE_STATE

30 **ResetRateControlCounter**

To reset the bit allocation accumulation counter to 0 to restart the rate control.

Value	Name
0	Not Reset
1	Reset

29:28 **RC Triggle Mode**

Value	Name	Description
00b	Always Rate Control	Whereas RC becomes active if sum_act > sum_target or sum_act < sum_target
01b	Gentle Rate Control	whereas RC becomes active if sum_act > upper_midpt or sum_act < lower_midpt
10b	Loose Rate Control	whereas RC becomes active if sum_act > sum_max or sum_act < sum_min
11b	Reserved	

27:24 **RC Stable Tolerance**

Format:	U4

This field specifies the tolerance required to deactivate RC once it has been triggered.

Value	Name
0-15	

23 **RC Panic Enable**

If this field is set to 1, RC enters panic mode when sum_act > sum_max. RC Panic Type field controls what type of panic behavior is invoked.

Value	Name
0	Disable
1	Enable

22 **RC Panic Type**

This field selects between two RC Panic methods

Value	Name
0	QP Panic
1	CBP Panic

Programming Notes
If it is set to 0, in panic mode, the macroblock QP is maxed out, setting to requested QP + QP_max_pos_mod. If it is set to 1, for an intra macroblock, AC CBPs are set to zero (note that DC CBPs are not modified). For inter macroblocks, AC and DC CBPs are forced to zero.

21 **MB Type Direct Conversion Disable**

Exists If:	//B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

MFX_AVC_SLICE_STATE

Value	Name
0	Enable direct mode conversion
1	Disable direct mode conversion

Programming Notes
This field is zero for all other slices other than B-Slice.

20 **MB Type Skip Conversion Disable**

Exists If:	//P-Slice or B-Slice

For all Macroblock type conversions in different slices, refer to Section "Macroblock Type Conversion Rules" in the same volume.

Value	Name
0	Enable skip type conversion
1	Disable skip type conversion

Programming Notes
This field is zero for all other slices other than P_Slice or B-Slice. \

19 **Is Last Slice**

It is used by the zero filling in the Minimum Frame Size test.

Value	Name	Description
1		Current slice is the last slice of a picture
0		Current slice is NOT the last slice of a picture

18 **Reserved**

17 **Header Insertion Present in Bitstream**

Value	Name	Description
0		No header insertion into the output bitstream buffer, in front of the current slice encoded bits.
1		Header insertion into the output bitstream buffer is present, and is in front of the current slice encoded bits.

Programming Notes
Note: In VDEnc mode, the slice header PAK object maximum size is 25 DWs.

16 **SliceData Insertion Present in Bitstream**

Value	Name	Description
0		No Slice Data insertion into the output bitstream buffer
1		Slice Data insertion into the output bitstream buffer is present.

15 **Tail Insertion Present in bitstream**

Value	Name	Description
0		No tail insertion into the output bitstream buffer, after the current slice

MFX_AVC_SLICE_STATE

		<table><tr><td></td><td></td><td>encoded bits</td></tr><tr><td>1</td><td></td><td>Tail insertion into the output bitstream buffer is present, and is after the current slice encoded bits.</td></tr></table>
	14	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
	13	**EmulationByteSliceInsertEnable** To have PAK outputting SODB or EBSP to the output bitstream buffer <table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>outputting RBSP</td></tr><tr><td>1</td><td></td><td>outputting EBSP</td></tr></table>
	12	**CabacZeroWordInsertionEnable** To pad the end of a SliceLayer RBSP to meet the encoded size requirement. <table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0</td><td></td><td>No Cabac_Zero_Word Insertion</td></tr><tr><td>1</td><td></td><td>Allow internal Cabac_Zero_Word generation and append to the end of RBSP (effectively can be used as an indicator for last slice of a picture, if the assumption is only the last slice of a picture needs to insert CABAC_ZERO_WORDs.</td></tr></table>
	11:8	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
	7:4	**Slice ID [3:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
	3:2	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
	1:0	**Stream ID [1:0]** To identify the output data (coding information record) returned for rate control from PAK to ENC and VPP.
7 Encoder Only	31:29	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
	28:0	**Indirect PAK-BSE Data Start Address (Write)** <table><tr><td>Exists If:</td><td>//AVC Encode Mode</td></tr></table> This field specifies the memory starting address (offset) to write out the compressed bitstream data from the BSE processing. This pointer is relative to the MFC Indirect PAK-BSE Object Base Address. It is a byte-aligned address for the AVC bitstream data in both CABAC/CAVLC Modes. For Write, there is no need to have a data length field. It is assumed the global memory bound check specified in the IND_OBJ_BASE_ADDRESS command (Indirect PAK-BSE Object Access Upper Bound) will take care of any illegal write access.

MFX_AVC_SLICE_STATE

		Value / **Name**: 0 - 512MB
8 Encoder Only	31:24	**Magnitude of QP Max Negative Modifier** Format: U8 This field specifies the lower limit of the QP modifier. **Value** / **Name**: 0-51
	23:16	**Magnitude of QP Max Positive Modifier** Format: U8 This field specifies the upper limit of the QP modifier. **Value** / **Name**: 0 - 15
	15:12	**Shrink Param - Shrink Resistance** Format: U4 This field specifies the additional points added each time decreased correction is invoked. **Value** / **Name**: 0 - 15
	11:8	**Shrink Param - Shrink Init** Format: U4 This field specifies the initial points required to trip decreased control. **Value** / **Name**: 0 - 15
	7:4	**Grow Param - Grow Resistance** Format: U4 This field specifies the additional points added each time increased correction is invoked. **Value** / **Name**: 0 - 15
	3:0	**Grow Param - Grow Init** Format: U4 This field specifies the initial points required to trip increased control. **Value** / **Name**: 0 - 15
9 Encoder Only	31	**RoundInterEnable** Format: Enable When this bit is not set, RoundInter defaults to 2.

MFX_AVC_SLICE_STATE

30:28 **RoundInter**

Format:	U3

Rounding precision for Inter quantized coefficients

Value	Name
000b	+1/16 **[Default]**
001b	+2/16
010b	+3/16
011b	+4/16
100b	+5/16
101b	+6/16
110b	+7/16
111b	+8/16

27 **RoundIntraEnable**

Format:	Enable

When this bit is not set, RoundIntra defaults to 4.

26:24 **RoundIntra**

Format:	U3

Rounding precision for Intra quantized coefficients

Value	Name
000b	+1/16 **[Default]**
001b	+2/16
010b	+3/16
011b	+4/16
100b	+5/16
101b	+6/16
110b	+7/16
111b	+8/16

23:20 **Correct 6**

Format:	U4

This field specifies the points used in the lowermost RC region when sum_act <= sum_min.

Value	Name
0 - 15	

19:16 **Correct 5**

Format:	U4

This field specifies the points used in the fifth RC region when sum_act > sum_min but <= lower_midpt.

MFX_AVC_SLICE_STATE

		Value: 0 - 15 / Name:
	15:12	**Correct 4** Format: U4 This field specifies the points used in the fourth RC region when sum_act > lower_midpt but <= sum_target. Value: 0 - 15 / Name:
	11:8	**Correct 3** Format: U4 This field specifies the points used in the third RC region when sum_act > sum_target but <= upper_midpt. Value: 0 - 15 / Name:
	7:4	**Correct 2** Format: U4 This field specifies the points used in the second RC region when sum_act > upper_midpt but <= sum_max. Value: 0 - 15 / Name:
	3:0	**Correct 1** Format: U4 This field specifies the points used in the topmost RC region when sum_act > sum_max. Value: 0 - 15 / Name:
10 Encoder Only	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**
	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0** Format: U4 If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2^{CV0}-1, they are replaced with 2^{CV0}-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped).

MFX_AVC_SLICE_STATE

For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:

none	CV7	CV5	CV4
CV7	CV6	CV4	CV3
CV5	CV4	CV2	CV1
CV4	CV3	CV1	CV0

For 8x8 frame block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV7	CV6	CV5	CV4	CV3	CV3
none	CV7	CV6	CV5	CV4	CV3	CV3	CV2
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV2
CV6	CV5	CV4	CV3	CV3	CV2	CV2	CV1
CV5	CV4	CV3	CV3	CV2	CV2	CV1	CV1
CV4	CV3	CV3	CV2	CV2	CV1	CV1	CV0
CV3	CV3	CV2	CV2	CV1	CV1	CV0	CV0
CV3	CV2	CV2	CV1	CV1	CV0	CV0	CV0

For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:

none	CV6	CV3	CV1
CV7	CV6	CV3	CV1
CV5	CV4	CV2	CV0
CV5	CV4	CV2	CV0

For 8x8 field block, each coefficient is mapped to one of the eight CV values as following:

none	none	CV6	CV5	CV4	CV3	CV2	CV1
none	CV7	CV6	CV5	CV4	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV3	CV2	CV1
CV7	CV6	CV5	CV4	CV3	CV2	CV2	CV1
CV6	CV5	CV4	CV4	CV3	CV2	CV1	CV0
CV6	CV5	CV4	CV3	CV2	CV2	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0
CV5	CV5	CV4	CV3	CV2	CV1	CV1	CV0

Value	Name
0 - 15	

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_WEIGHTOFFSET_STATE

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

This is a slice level command and can be issued multiple times within a picture that is comprised of multiple slices. The same command is used for AVC encoder (PAK mode) and decoder (VLD and IT modes). However, since for AVC decoder VLD and IT modes, and AVC encoder mode, the implicit weights are computed in hardware, this command is not issued. For encoder, regardless of the type of weight calculation is active for the current slice (default, implicit or explicit), they are all sent to the PAK as if they were all in explicit mode. However, for implicit weight and offset, each entry contains only a 16-bit weight and no offset (offset = 0 always in implicit mode and can be hard-coded inside the hardware).The weights (and offsets) are needed in processing both P and B slice in AVC codec. For P-MB, at most only L0 list is used; for B-MB both L0 and L1 lists may be needed. For a B-MB that is coded in L1-only Prediction, only L1 list is sent.The content of this command matches with the DXVA2 AVC API data structure for explicit prediction mode only : Weights[2][32][3][2] (L0:L1, 0:31 RefPic, Y:Cb:Cr, W:0)

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h PARALLEL_VIDEO_PIPE
		Format:	OpCode
	28:27	**Pipeline**	
		Default Value:	2h MFX_ AVC_ WEIGHTOFFSET_STATE
		Format:	OpCode
	26:24	**Media Command Opcode**	
		Default Value:	1h AVC_COMMON
		Format:	OpCode
	23:21	**SubOpcode A**	
		Default Value:	0h
		Format:	OpCode
	20:16	**SubOpcode B**	
		Default Value:	5h
		Format:	OpCode
	15:12	**Reserved**	
		Format:	MBZ
	11:0	**DWord Length**	
		Default Value:	60h Excludes DWord (0,1)
		Format:	=n Total Length - 2

MFX_AVC_WEIGHTOFFSET_STATE

1	31:1	**Reserved** Format: MBZ
	0	**Weight and Offset Select** It must be set in consistent with the WeightedPredFlag and WeightedBiPredIdc in the Img_State command. This parameter is specified for Intel interface only, not present in the DXVA. For implicit even though only one entry may be used, still loading the whole 32-entry table.

Value	Name	Description
0	Weight and Offset L0 table	The list that followed is associated with the weight and offset for RefPicList L0
1	Weight and Offset L1 table	The list that followed is associated with the weight and offset for RefPicList L1

2..97	31:0	**WeightOffset** WeightOffset[L=L0=0 or L1=1][i=0 to 31][Y=0/Cb=1/Cr=2][weight=0/offset=1] WeightOffset[L][i=0][Y=0][Weight=0], WeightOffset[L][i=0][Y=0][Offset=1] WeightOffset[L][i=0][Cb=1][Weight=0], WeightOffset[L][i=0][Cb=1][Offset=1] WeightOffset[L][i=0][Cr=2][Weight=0], WeightOffset[L][i=0][Cr=2][Offset=1]: WeightOffset[L][i=31][Y=0][Weight=0], WeightOffset[L][i=31][Y=0][Offset=1] WeightOffset[L][i=31][Cb=1][Weight=0], WeightOffset[L][i=31][Cb=1][Offset=1] WeightOffset[L][i=31][Cr=2][Weight=0], WeightOffset[L][i=31][Cr=2][Offset=1] Format for explicit: Both Weight and Offset are S15 in two's compliment, with a valid range from -128 to 128 Format for implicit: S15

MFX_AVC_WEIGHTOFFSET_STATE		
		This set of fields is always present whenever this command is issued. The full table, one entry for each reference picture, is always specified. Any reference list L0/L1[i] that does not exist, the corresponding weight and offset are set to 0. Weight and Offset are 2 byte each. Apair of Weight and Offset forms a dword, with Weight in the LOWER word and Offset in the HIGHER word. WeightOffset[L0=0][i=0 to 31][Y=0] (i.e. luma_weight_l0[i]) are specified for the weighting and offset factors applied to the luma prediction value for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When luma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of luma_weight_l0[i] shall be in the range of -128 to 127. When luma_weight_l0_flag is equal to 0, luma_weight_l0[i] shall be inferred to be equal to 2luma_log2_weight_denom for RefPicList0[i]. luma_log2_weight_denom is a Slice Header syntax element. WeightOffset[L0=0][i=0 to 31][Cb=1] (i.e. chromaCb_weight_l0[i]) are specified for the weighting and offset factors applied to the chroma Cb prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of chromaCb_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCb_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i]. chroma_log2_weight_denom is a Slice Header syntax element. WeightOffset[L0=0][i=0 to 31][Cr=2] (i.e. chromaCr_weight_l0[i]) are specified for the weighting and offset factors applied to the chroma Cr prediction values for list 0 prediction using RefPicList0[i] (one-to-one correspondence in i). When chroma_weight_l0_flag (Slice Header syntax element) is equal to 1, the value of chromaCr_weight_l0[i] shall be in the range of -128 to 127. When chroma_weight_l0_flag is equal to 0, chromaCr_weight_l0[i] shall be inferred to be equal to 2chroma_log2_weight_denom for RefPicList0[i].

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This frame-level state command is used to specify all the buffer base addresses needed for the operation of the AVC Bit Stream Processing Units (for decoder, it is BSD Unit; for encoder, it is BSE Unit) For both encoder and decoder, currently it is assumed that all codec standards can share the same BSP_BUF_BASE_STATE. The simplicity of this command is the result of moving all the direct MV related processing into the ENC Subsystem. Since all implicit weight calculations and directMV calculations are done in ENC and all picture buffer management are done in the Host, there is no need to provide POC (POC List - FieldOrderCntList, CurrPic POC - CurrFieldOrderCnt) information to PAK. For decoder, all the direct mode information are sent in a separate slice-level command (AVC_DIRECTMODE_STATE command). In addition, in Encoder, the row stores for CABAC encoding and MB Parameters Construction (MPC) are combined into one single row store. The row stores specified in this command do not combine with those specified in the MFC_PIPE_BUF_ADDR_STATE command for hardware simplification reason.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Pipeline Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 4h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 8h Excludes DWord (0,1)

MFX_BSP_BUF_BASE_ADDR_STATE

		Project: All Format: =n Total Length - 2
1	31:6	**BSD/MPC Row Store Scratch Buffer Base Address - Read/Write** This field provides the base address of the scratch buffer used by BSD (decoder) and MPC (encoder) unit to store MB information of the previous row for coding each macroblock in the current row. It is a private buffer used by the BSD (decoder) and MPC (encoder) hardware only. Its content is not accessible by software. ThisRow Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address this Row Store. For AVC BSD, 2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF. So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes = 32,768 bytes are required. Cacheline alignment should be followed. For AVC MPC, 1 cachline for non-MBAFF, 2 cachelines for MBAFF per MB. For VC1, the BSD row store is 512-bit (one cacheline) per MB, times the number of MBs per picture MB row. **Programming Notes** This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cacheline address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage. *(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).*
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**BSD/MPC Row Store Scratch Buffer Base Address - Read/Write [47:32]** Project: CHV, BSW **Description** This field is for the upper range of BSD/MPC Row Store Scratch Buffer Base Address. This field is used for 48-bit addressing.
3 **Project:**	31:15	**Reserved**
	14:13	**Reserved**

MFX_BSP_BUF_BASE_ADDR_STATE

CHV, BSW

12 — BSD/MPC Row Store Scratch Buffer Cache Select

Project:	CHV, BSW

This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC.

Value	Name	Description
0		Buffer going to LLC
1		Buffer going to Internal Media Storage

11 — Reserved

Project:	CHV, BSW
Format:	MBZ

10:9 — Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 — BSD/MPC Row Store Scratch Buffer - Arbitration Priority Control

Project:	CHV, BSW
Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 — Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for BSD/MPC Row Store Scratch Buffer Base Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 — Target Cache (TC) for BSD/MPC Row Store Scratch Buffer Base Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT

MFX_BSP_BUF_BASE_ADDR_STATE

		01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) BSD/MPC Row Store Scratch Buffer Base Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
4	31:6	**MPR Row Store Scratch Buffer Base Address - Read/Write (Decoder Only)** This field provides the base address of the scratch buffer used by decoder's MPR unit to store MB information of the previous row for decoding each macroblock in the current row. It is a private buffer used by the MPR hardware only. Its content is not accessible by software. **Programming Notes** The MPR Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of each macroblock to address the MPR Row Store. Except ILDB Control Data, all other operations does not cross slice boundary. This field is specified in frame-level.2 cacheline (CL) per MB when in MBAFF mode (row of MB pair); 1 CL per MB for non-MBAFF, So, to support 256 MBs per row (4K screen resolution), 2 * 256 * 64 bytes = 32,768 bytes are required. Cacheline alignment should be followed.This field is only valid for AVC decoder mode This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Storage (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be cache inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage *(Notes: 1 cachelines per MB for non-mbaff; 2 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).*
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
5 **Project:** CHV, BSW	31:16	**Reserved** Project: All Format: MBZ Reserved for 64-bit address extension.

MFX_BSP_BUF_BASE_ADDR_STATE

	15:0	**MPR Row Store Scratch Buffer Base Address - Read/Write [47:32]** Project: All This field is for the upper range of MPR Row Store Scratch Buffer Base Address. This field is used for 48-bit addressing.
6 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**MPR Row Store Scratch Buffer Cache Select** Project: CHV, BSW This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC. **Value** / **Name** / **Description** 0 / Buffer going to LLC 1 / Buffer going to Internal Media Storage
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**MPR Row Store Scratch Buffer - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b / Highest priority 01b / Second highest priority 10b / Third highest priority 11b / Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MPR Row Store Scratch Buffer Base Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be

MFX_BSP_BUF_BASE_ADDR_STATE

generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 Target Cache (TC) MPR Row Store Scratch Buffer Base Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

1:0 Age for QUADLRU (AGE) MPR Row Store Scratch Buffer Base Address

Project:	CHV, BSW
Format:	Enable

This field allows the selection of AGE parameter for a given surface in LLC or eLLC.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

7 — 31:6 Bitplane Read Buffer Base Address

Project:	All

It must be cacheline aligned (i.e. 64 bytes address boundary), so lower bit 0 to 5 are used for controlling information. Bitplane buffer is a linear buffer. In VC1 Long format, it is written by an application. In VC1 Short Format, it is written and read by H/W only.For VC1 intel Long Format : it is a read-only bufferFor VC1 DXVA2 Short Format : it is a write and a read bufferThis field is only valid for VC1 decoder mode.

5:0 Reserved

Project:	CHV, BSW
Format:	MBZ

8 — 31:16 Reserved

Project:	CHV, BSW

Project:

MFX_BSP_BUF_BASE_ADDR_STATE

CHV, BSW		Format: MBZ Reserved for 64-bit address extension.
	15:0	**Bitplane Read Buffer Base Address - Read/Write [47:32]** Project: All This field is for the upper range of Bitplane Read Buffer Base Address. This field is used for 48-bit addressing.
9 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Bitplane Read Buffer - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Bitplane Read Buffer Base Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable

MFX_BSP_BUF_BASE_ADDR_STATE

		10b: Writethrough (WT) 11b: Writeback (WB)
	4:3	**Target Cache (TC) Bitplane Read Buffer Base Address** Project: CHV, BSW This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC 00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)* 01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*). 10b: LLC/eLLC Allowed. 11b: L3, LLC, eLLC Allowed. **Errata CHV:A-E (FIXED BY:G0 Stepping):** For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching. For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops. Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed. **Value** / **Name** 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) Bitplane Read Buffer Base Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("0,1,2") it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. **Value** / **Name** 11b: Good chance of generating hits 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits

MFX_DBK_OBJECT

MFX_DBK_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_DBK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h Common Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0Bh Excludes DWord (0,1) Format: =n Note: Regardless of the mode, inline data must be present in this command
1	31:6	**Pre Deblocking Source Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit).
	5:0	**Reserved** Project: CHV, BSW Format: MBZ

MFX_DBK_OBJECT

2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Pre Deblocking Source Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Pre-Deblocking Source Address. This field is used for 48-bit addressing.
3 **Project:** CHV, BSW	31:15	**Reserved**
	14:13	**Reserved**
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Pre Deblocking Source - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Pre Deblocking Source Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** 00b: Use Cacheability Controls from page table 01b: Uncacheable (UC) - non-cacheable 10b: Writethrough (WT) 11b: Writeback (WB)

MFX_DBK_OBJECT

	4:3	**Target Cache (TC) for Pre Deblocking Source Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Pre Deblocking Source Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
4	31:6	**Deblocking Control Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address as input MB-level deblocking parameters to control the way hardware deblock the each micro-block. One 512-bit cacheline is allocated for each Macroblock in raster scan order.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
5 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Deblocking Control Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Control Address (DeblockCntrlAddr). This field is used for 48-bit addressing.
6 **Project:** CHV, BSW	31:15	**Reserved** Format: MBZ
	14:13	**Reserved**

MFX_DBK_OBJECT

	Project:	CHV, BSW
	Format:	MBZ

12:11 Reserved

Project:	CHV, BSW
Format:	MBZ

10:9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 Deblocking Control - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Deblocking Control Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 Target Cache (TC) for Deblocking Control Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

1:0 Age for QUADLRU (AGE) for Deblocking Control Address

Project:	CHV, BSW

MFX_DBK_OBJECT

		Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. **Value** / **Name** 11b – Good chance of generating hits. 10b – Next good chance of generating hits 01b – Decent chance of generating hits 00b – Poor chance of generating hits
7	31:6	**Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit)
	5:0	**Reserved**
8 **Project:** CHV, BSW	31:16	**Reserved**
	15:0	**Deblocking Destination Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Deblocking Destination Address. This field is used for 48-bit addressing.
9 **Project:** CHV, BSW	31:15	**Reserved**
	14:13	**Reserved**
	12:11	**Reserved**
	10:9	**Reserved**
	8:7	**Deblocking Destination - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Deblocking Destination Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name**

MFX_DBK_OBJECT

		00b: Use Cacheability Controls from page table 01b: Uncacheable (UC) - non-cacheable 10b: Writethrough (WT) 11b: Writeback (WB)
	4:3	**Target Cache (TC) for Deblocking Destination Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Deblocking Destination Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. 11b: Good chance of generating hits 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
10	31:6	**Deblock Row Store Address** Format: GraphicsAddress[31:6] This field provides the base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of the current macroblock to address the Deblocking Filter Row Store. **Programming Notes** This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Cache (total size 640 CacheLine). When Deblocking Row Store Cache Select is programmed to 1, this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage. Also, this command is only for Standalone Deblocker Mode. During Standalone Deblocker Mode, only this Deblock Row Store will be active. There is no other row store active so it can use the whole internal Media Cache. (Notes: 2 cachelines per MB for non-mbaff; 4 cachelines per MB pair for mbaff, and the buffer needs to have enough space for 1

MFX_DBK_OBJECT

		MB (pair) row.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
11 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**Deblock Row Store Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Deblock Row Store Address (DeblockRowStoreAddr). This field is used for 48-bit addressing.
12 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**Deblock Row Store - Cache Select** Project: CHV, BSW This field controls if Deblock Row Store is going to store inside Media Cache or to LLC. **Value** / **Name** 0 – Buffer going to LLC. 1 – Buffer going to Internal Media Storage
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Deblock Row Store - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name**

MFX_DBK_OBJECT

00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Deblock Row Store Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 Target Cache (TC) for Deblock Row Store Address

Project:	CHV, BSW

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC
00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)*
01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*).
10b: LLC/eLLC Allowed.
11b: L3, LLC, eLLC Allowed.
Errata CHV:A-E (FIXED BY:G0 Stepping):
For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching.
For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops.
Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

MFX_DBK_OBJECT

	1:0	**Age for QUADLRU (AGE) for Deblock Row Store Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age (“0,1,2”) it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

MFX_FQM_STATE

MFX_FQM_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices.This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B).In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 8h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:2	**Reserved** Format: MBZ

MFX_FQM_STATE

	1:0	**AVC** Exists If: //AVC- Decoder Only **For AVC QM Type**: This field specifies which Quantizer Matrix is loaded. 0 – AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 1 – AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 2 – AVC_8x8_Intra_MATRIX 3 – AVC_8x8_Inter_MATRIX
	1:0	**MPEG2** Exists If: //MPEG2- Decoder Only **For MPEG2 QM Type**: This field specifies which Quantizer Matrix is loaded. 0 – MPEG_INTRA_QUANTIZER_MATRIX 1 – MPEG_NON_INTRA_QUANTIZER_MATRIX 2-3 – Reserved
	1:0	**JPEG** Project: CHV, BSW Exists If: //JPEG- Encoder Only **For JPEG QM Type**:This field specifies which Quantizer Matrix is loaded. 0 – JPEG_Luma_Y_QUANTIZER_MATRIX (or R) 1 – JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G) 2 – JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B) **Programming Notes** For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j]. In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.
2..33	31:0	**Forward Quantizer Matrix** Project: All Format: U32 The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers). This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculated the corresponding memory location within the frame buffer directly.

The MFX_IND_OBJ_BASE_ADDR command sets the memory base address pointers for the corresponding Indirect Object Data Start Addresses (Offsets) specified in each OBJECT commands. The characteristic of these indirect object data is their variable size (per MB or per Slice). Hence, each OBJECT command must specify the indirect object data offset from the base address to start fetching or writing object data.

While the use of base address is unconditional, the indirection can be effectively disabled by setting the base address to zero.
For decoder, there are:

- 1 read-only per-slice indirect object in the BSD_OBJECT Command, and
- 2 read-only per-MB indirect objects in the IT_OBJECT Command.

For decoder: the Video Command Streamer (VCS) will perform the memory access bound check automatically using the corresponding MFC Indirect Object Access Upper Bound specification. If any access is at or beyond the upper bound, zero value is returned. The request to memory is still being sent, but the corresponding codec's BSD unit will detect this condition and perform the zeroing return. If the Upper Bound is turned off, the beyond bound request will return whatever on the bus (invalid data).
For encoder, there are:

- 1 read-only per-MB indirect object in the PAK_OBJECT Command, and
- 1 write-only per-slice indirect object in the PAK Slice_State Command

For encoder: whenever an out of bound address accessing request is generated, VMX will detect such requests and snap the address to the corresponding [indirect object base address + indirect data start address]. VMX will return all 0s as the data to the requestor. NotationDefinitionPhysicalAddress[n:m] Corresponding bits of a physical graphics memory byte address (not mapped by a GTT) GraphicsAddress[n:m] Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode

MFX_IND_OBJ_BASE_ADDR_STATE

	28:27	**Pipeline** Default Value: 2h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	26:24	**Common Opcode** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	23:21	**Sub OpcodeA** Default Value: 0h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	20:16	**SubOpcodeB** Default Value: 3h MFX_IND_OBJ_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0018h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**MFX Indirect Bitstream Object - Base Address (Decoder and Stitch Modes)** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_XXX_BSD_OBJECT command for fetching (reading) the compressed Slice Data.This field is only valid in MPEG2, AVC and VC1 decoder VLD mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31:16	**Reserved** Project: All Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect Bitstream Object - Destination Address (Decoder and Stitch Modes)[47:32]** Project: All

MFX_IND_OBJ_BASE_ADDR_STATE

		Description
		This field is for the upper range of MFX Indirect Bitstream Object Base Address.
		This field is used for 48-bit addressing.
3	31:15	**Reserved**
		Project: CHV, BSW
		Format: MBZ
	14:13	**Reserved**
		Project: CHV, BSW
		Format: MBZ
	12:11	**Reserved**
		Project: CHV, BSW
		Format: MBZ
	10:9	**Reserved**
		Project: CHV, BSW
		Format: MBZ
	8:7	**MFX Indirect Bitstream ObjectBase - Arbitration Priority Control**
		Project: CHV, BSW
		Format: U2
		This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
		Value / **Name**
		00b: Highest priority
		01b: Second highest priority
		10b: Third highest priority
		11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MFX Indirect Bitstream ObjectBase Address**
		Project: CHV, BSW
		This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.
		Value / **Name**
		00b: Use Cacheability Controls from page table
		01b: Uncacheable (UC) - non-cacheable
		10b: Writethrough (WT)
		11b: Writeback (WB)
	4:3	**Target Cache (TC) MFX Indirect Bitstream ObjectBase Address**
		Project: CHV, BSW

MFX_IND_OBJ_BASE_ADDR_STATE

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 — **Reserved**

1:0 — **Age for QUADLRU (AGE) MFX Indirect Bitstream ObjectBase Address**

Project:	CHV, BSW
Format:	Enable

This field allows the selection of AGE parameter for a given surface in LLC or eLLC.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

4, 31:12 — **MFX Indirect Bitstream Object - Access Upper Bound (Decoder and Stitch Modes)**

Format:	GraphicsAddress[31:12]

This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_XXX_BSD_OBJECT command for the compressed Slice Data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFX Indirect Bitstream ObjectBase Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFD_XXX_BSD_OBJECT command is set to 0.This field is only valid in MPEG2, AVC, VP8, and VC1 decoder VLD mode.

Programming Notes	Project
For **VP8 Encoder**, this field is corresponding to **MFC Indirect PAK-BSE Object - Access Upper Bound in DW24, DW25**. Please program Indirect bitstream upperbound in this field the same as DW24, DW25.	CHV, BSW

11:0 — **Reserved**

Format:	MBZ

5, 31:16 — **Reserved** — **Project:** CHV, BSW

Project:	CHV, BSW
Format:	MBZ

Reserved for 64-bit address extension.

15:0 — **MFX Indirect Bitstream Object UpperBound (Decoder and Stitch Modes)[47:32]**

Project:	CHV, BSW

MFX_IND_OBJ_BASE_ADDR_STATE

Description	Project
This field is for the upper range of MFX Indirect Bitstream Object UpperBound.	
This field is used for 48-bit addressing.	CHV, BSW

Programming Notes	Project
For **VP8 Encoder**, this field is corresponding to **MFC Indirect PAK-BSE Object - Access Upper Bound in DW24, DW25**. Please program Indirect bitstream upperbound in this field the same as DW24, DW25.	CHV, BSW

6 — 31:12 — MFX Indirect MV Object - Base Address

Format:	GraphicsAddress[31:12]

Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the encoder MFC_AVC_PAK_OBJECT command or the decoder MFD_IT_OBJECT command for fetching the per-MB MV data.This field is only valid in AVC encoder mode or in AVC decoder IT mode

11:6 — Reserved

Format:	MBZ

5:0 — Reserved

Project:	CHV, BSW
Format:	MBZ

7 — Project: CHV, BSW

31:16 — Reserved

Project:	CHV, BSW
Format:	MBZ

Reserved for 64-bit address extension.

15:0 — MFX Indirect MV Object Base Address [47:32]

Project:	All

Description	Project
This field is for the upper range of MFX Indirect MV Object Base Address.	
This field is used for 48-bit addressing.	CHV, BSW

8 — Project: CHV, BSW

31:15 — Reserved

Project:	CHV, BSW
Format:	MBZ

14:13 — Reserved

Project:	CHV, BSW
Format:	MBZ

MFX_IND_OBJ_BASE_ADDR_STATE

	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**MFX Indirect MV Object - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value / Name: 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MFX Indirect MV ObjectBase Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. Value / Name: 00b – Use Cacheability Controls from page table 01b – Uncacheable (UC) - non-cacheable 10b – Writethrough (WT) 11b – Writeback (WB)
	4:3	**Target Cache (TC) MFX Indirect MV ObjectBase Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching Value / Name: 00b – eLLC Only - not snooped in GT 01b – LLC Only 10b – LLC/eLLC Allowed 11b – L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) MFX Indirect MV ObjectBase Address** Project: CHV, BSW

MFX_IND_OBJ_BASE_ADDR_STATE

		Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. Value / Name: 11b – Good chance of generating hits. 10b – Next good chance of generating hits 01b – Decent chance of generating hits 00b – Poor chance of generating hits
9	31:12	**MFX Indirect MV Object Access Upper Bound** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command for the per-MB MV data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFX Indirect MV Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Data Length field of the MFC_AVC_PAK_OBJECT / MFD_IT_OBJECT command is set to 0.This field is only valid in AVC encoder mode or in AVC decoder IT mode.
	11:0	**Reserved** Format: MBZ
10 **Project:** CHV, BSW	31:16	**Reserved** Project: All Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFX Indirect MV Object UpperBound [47:32]** Project: All Description / Project: This field is for the upper range of MFX Indirect MV Object Base Address. – This field is used for 48-bit addressing. – CHV, BSW
11	31:12	**MFD Indirect IT-COEFF Object - Base Address (Decoder Only)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB non-scaled coefficient data (all inverse scaling and quantization are done in hardware).This field is only valid in MPEG2, AVC and VC1 decoder IT mode.
	11:6	**Reserved**

MFX_IND_OBJ_BASE_ADDR_STATE

		Format: MBZ
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
12 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-COEFF Object Base Address [47:32]** Project: CHV, BSW

Description	Project
This field is for the upper range of MFX Indirect IT-COEFF Object Base Address.	
This field is for the upper range of MFX Indirect MV Object Base Address.	
This field is used for 48-bit addressing.	CHV, BSW

13 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved**
	8:7	**MFD Indirect IT-COEFF Object Desitnation - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MFD Indirect IT-COEFF ObjectBase Address**

MFX_IND_OBJ_BASE_ADDR_STATE

		Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** 00b: Use Cacheability Controls from page table 01b: Uncacheable (UC) - non-cacheable 10b: Writethrough (WT) 11b: Writeback (WB)
	4:3	**Target Cache (TC) MFD Indirect IT-COEFF ObjectBase Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching **Value** / **Name** 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) MFD Indirect IT-COEFF ObjectBase Address** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. **Value** / **Name** 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
14	31:12	**MFD Indirect IT-COEFF Object - Access Upper Bound (Decoder Only)** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB non-scaled coefficient data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFD Indirect IT-COEFF Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect COEFF Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in MPEG2, AVC and VC1 decoder IT mode.
	11:0	**Reserved**

MFX_IND_OBJ_BASE_ADDR_STATE

		Format: MBZ
15 **Project:** CHV, BSW	31:16	**Reserved** Project: All Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-COEFF Object UpperBound [47:32]** Project: CHV, BSW **Description** This field is for the upper range of MFX Indirect IT-COEFF Object UpperBound. This field is for the upper range of MFX Indirect MV Object Base Address. This field is used for 48-bit addressing.
16	31:12	**MFD Indirect IT-DBLK Object - Base Address (Decoder Only)** Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the read-only indirect data object pointed in the MFD_IT_OBJECT command for fetching (reading) the per-MB Deblocking filter control data.This field is only valid in AVC decoder IT mode.
	11:6	**Reserved** Format: MBZ
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
17 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-DBLK Object Base Address [47:32]** Project: CHV, BSW **Description** / **Project** This field is for the upper range of MFX Indirect IT-DBLK Object Base Address. / This field is used for 48-bit addressing. / CHV, BSW
18 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved**

MFX_IND_OBJ_BASE_ADDR_STATE

	Project:	CHV, BSW
	Format:	MBZ

12:11 Reserved

Project:	CHV, BSW
Format:	MBZ

10:9 Reserved

8:7 MFD Indirect IT-DBLK Object - Arbitration Priority Control

Project:	CHV, BSW
Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MFD Indirect IT-DBLK ObjectBase Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 Target Cache (TC) MFD Indirect IT-DBLK ObjectBase Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

1:0 Age for QUADLRU (AGE) MFD Indirect IT-DBLK ObjectBase Address

MFX_IND_OBJ_BASE_ADDR_STATE

		Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. **Value** / **Name** 11b: Good chance of generating hits 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
19	31:12	**MFD Indirect IT-DBLK Object - Access Upper Bound (Decoder Only)** Format: GraphicsAddress[31:12] This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the MFD_IT_OBJECT command for the per-MB Deblocking filter control data. Indirect data accessed at this address and beyond will return as 0 by the hardware. Setting this field to 0 will cause this range check to be ignored.If non-zero, this address must be greater than the MFD Indirect IT-DBLK Object Base Address state.Hardware ignores this field if indirect data is not present, i.e. the Indirect Deblocking Control Data Length field of the MFD_IT_OBJECT command is set to 0.This field is only valid in AVC decoder IT mode.
	11:0	**Reserved** Format: MBZ
20 **Project:** CHV, BSW	31:16	**Reserved** Project: All Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFD Indirect IT-DBLK Object UpperBound [47:32]** Project: All **Description** / **Project** This field is for the upper range of MFX Indirect IT-DBLK Object UpperBound. / This field is used for 48-bit addressing. / CHV, BSW
21	31:12	**MFC Indirect PAK-BSE Object - Base Address (Encoder Only)** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned memory base address for the write-only indirect data object pointed in the PAK_SLICE_STATE command for writing out the compressed bitstream.This field is only valid in AVC encoder mode.
	11:6	**Reserved** Project: All

MFX_IND_OBJ_BASE_ADDR_STATE

		Format: MBZ
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
22 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**MFC Indirect PAK-BSE Object Base Address [47:32]** Project: CHV, BSW This field is for the upper range of MFX Indirect PAK-BSE Object Base Address. This field is used for 48-bit addressing. (Project: CHV, BSW)
23 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**MFC Indirect PAK-BSE Object Desitnation - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for MFC Indirect PAK-BSE**

MFX_IND_OBJ_BASE_ADDR_STATE

ObjectBase Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 **Target Cache (TC) MFC Indirect PAK-BSE ObjectBase Address**

Project:	CHV, BSW

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC
00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)*
01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*).
10b: LLC/eLLC Allowed.
11b: L3, LLC, eLLC Allowed.
Errata CHV:A-E (FIXED BY:G0 Stepping):
For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching.
For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops.
Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

1:0 **Age for QUADLRU (AGE) MFC Indirect PAK-BSE ObjectBase Address**

Project:	CHV, BSW
Format:	Enable

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a

MFX_IND_OBJ_BASE_ADDR_STATE

particular allocation is done at youngest age (“0,1,2”) it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

24 **Project:** CHV, BSW — 31:12

MFC Indirect PAK-BSE Object - Access Upper Bound (Eecoder Only)

Project:	CHV, BSW
Format:	GraphicsAddress[31:12]

This field specifies the 4K-byte aligned (exclusive) maximum Graphics Memory address access by the indirect data object in the PAK_SLICE_STATE command for the per-slice output bitstream. Indirect data accessed at this address and beyond will be blocked by the hardware and ignored. Setting this field to 0 will cause this range check to be ignoredIf non-zero, this address must be greater than the MFC Indirect PAK-BSE Object Base Address state.This field is only valid in AVC encoder mode.

Programming Notes	Project
For VP8 Encoder, this field should be programmed the same at both DW4, DW5 **MFX Indirect Bitstream Object - Access Upper Bound** as well as DW24, DW25.	CHV, BSW

11:0 — **Reserved**

25 **Project:** CHV, BSW — 31:16

Reserved

Project:	All
Format:	MBZ

Reserved for 64-bit address extension.

15:0 — **MFC Indirect PAK-BSE Object UpperBound [47:32]**

Project:	CHV, BSW

Description	Project
This field is for the upper range of MFC Indirect PAK-BSE Object UpperBound	
This field is used for 48-bit addressing.	CHV, BSW

Programming Notes	Project
For VP8 Encoder, this field should be programmed the same at both DW4, DW5 **MFX Indirect Bitstream Object - Access Upper Bound** as well as DW24, DW25.	CHV, BSW

MFX_JPEG_HUFF_TABLE_STATE

MFX_JPEG_HUFF_TABLE_STATE

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

This Huffman table commands contains both DC and AC tables for either luma or chroma. Once a Huffman table has been defined for a particular destination, it replaces the previous tables stored in that destination and shall be used in the remaining Scans of the current image. A Huffman table will be sent to H/W only when it is loaded from bitstream.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved**
	11:0	**DWord Length** Default Value: 033Dh Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:1	**Reserved**
	0	**HuffTableID (1-bit)** Identifies the huffman table. Value 0 – Name: Y – Description: Huffman table for Y

MFX_JPEG_HUFF_TABLE_STATE		
2..4	31:0	**DC_BITS (12 8-bit array)** The number of DC Huffman codes of length i, where i is 1~12
5..7	31:0	**DC_HUFFVAL (12 8-bit array)** The value associated with each DC Huffman code of length i.
8..11	31:0	**AC_BITS (16 8-bit array)** the list of Li, number of Huffman codes of length i, where i is 1~16
12..51	31:0	**AC_HUFFVAL (160 8-bit array)** the list of Vi,j, the value associated with each Huffman code of length i
52	31:16	**Reserved** Project: All Format: MBZ
	15:0	**AC_HUFFVAL(2-8 bit array)** In AC table, BITS can have up to 16-bit codeword. Li can be 0 ~ 162. HUFFVAL will have a list of likely random distributed values

MFX_JPEG_PIC_STATE

MFX_JPEG_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 7h JPEG Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Common Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h MEDIA_ Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Project: All Format: =n Total Length - 2 Value / Name / Description 0001h / **[Default]** / Excludes DWord (0,1)
1	31	**Reserved**
	30:26	**Pixels In Horizontal Last MCU** Project: CHV, BSW Exists If: //Encoder Only The number of pixels in the last MCU in a row MCUs. This information is used for completion of partial MCU.

MFX_JPEG_PIC_STATE

	31:21	**Reserved** Exists If: //Decoder Only Format: MBZ
	25:21	**Pixels In Vertical Last MCU** Project: CHV, BSW Exists If: //Encoder Only The number of pixels in the last MCU in a column MCUs. This information is used for completion of partial MCU.
	20	**Vertical Up-Sampling Enable** Project: CHV, BSW Exists If: //Decoder Only Only applied to chroma blocks. This flag is used for 2:1 vertical up-sampling for chroma 420 and outputting chroma422 YUY2 or UYVY format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV420, and **OutputFormatYUV** should be set to YUY2 or UYVY.

Value	Name	Description
0b		no up-sampling
1b		2:1 vertical up-sampling

	19	**Reserved** Project: CHV, BSW Exists If: //Decoder Only
	18	**Horizontal Down-Sampling Enable** Project: CHV, BSW Exists If: //Decoder Only Only applied to chroma blocks. This flag is used for 2:1 horizontal down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV422V_2Y or YUV422V_4Y, and **OutputFormatYUV** should be set to NV12.

Value	Name	Description
0b		no down-sampling
1b		2:1 horizonatl down-sampling

	17	**Vertical Down-Sampling Enable** Project: CHV, BSW Exists If: //Decoder Only Only applied to chroma blocks. This flag is used for 2:1 vertical down-sampling for chroma 422 and outputting chroma420 NV21 format. To enable this flag, the input should be interleaved Scan, **InputFormatYUV** should be set to YUV422H_2Y or YUV422H_4Y, and **OutputFormatYUV** should be set to NV12.

MFX_JPEG_PIC_STATE

Value	Name	Description
0b		no down-sampling
1b		2:1 vertical down-sampling

16 **Average Down Sampling**

Project:	CHV, BSW
Exists If:	//Decoder Only

This flag is used to select a down-sampling method when **VertDownSamplingEnb** or **HoriDownSamplingEnb** is set to 1.

Value	Name	Description
0b		Drop every other line (or column) pixels
1b		Average neighboring two pixels

20:12 **Reserved**

Project:	CHV, BSW
Exists If:	//Encoder Only
Format:	MBZ

15:12 **Reserved**

Exists If:	//Decoder Only
Format:	MBZ

11:8 **Output Format YUV**

Project:	CHV, BSW
Exists If:	//Decoder Only

This field specifies the surface format to write the decoded JPEG image.Note that any non-interleaved JPEG input should be set to "0000". For the interleaved input Scan data, it can be set either "0000" or the corresponding format.

Value	Name	Description
0000b		3 separate plane for Y, U, and V respectively
0001b		NV12 for chroma 4:2:0
0010b		UYVY for chroma 4:2:2
0011b		YUY2 for chroma 4:2:2

Programming Notes

The **MFX_SURFACE_STATE** command should be set accordingly for each **OutputFormatYUV**.
For NV12, **Surface Format** = 4 (PLANAR_420_8)
For YUY2, **Surface Format** = 0 (YCRCB_NORMAL)
For UYVY, **Surface Format** = 3 (YCRCB_SWAPY)
NV12 (0001b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases

- **InputFormatYUV** is YUV420 and **VertDownSamplingEnb** is disalbed

MFX_JPEG_PIC_STATE

- **InputFormatYUV**is YUV422H_2Y or YUV422H_4Y, and **VertDownSamplingEnb** is enabled

UYVY (0010b) and YUY2 (0011b) can be set only when Y, U, V are interleaved in a single Scan data with the following cases

- **InputFormatYUV** is YUV420 and **VertUpSamplingEnb** is enabled
- **InputFormatYUV** is YUV422H_2Y or YUV422H_4Y and **VertUpSamplingEnb** is disabled

11:8 **Input Surface Format YUV**

Project:	CHV, BSW
Exists If:	//Encoder Only

This field specifies the surface format to read a YUV image data

Value	Name	Description
0000b		Reserved
0001b	NV12	NV12 for chroma 4:2:0
0010b	UYVY	UYVY for chroma 4:2:2
0011b	YUY2	YUY2 for chroma 4:2:2
0100b	Y8	Y8 for chroma400 Y-only image
0101b	RGB	RGB or YUV for chroma 4:4:4

Programming Notes
This field should be set accordingly for **SurfaceFormat** in MFX_SURFACE_STATE command.
R8G8B8A8_UNORM in this field is used for encoding RGB and YUV chroma 4:4:4. For RGB input, any order of image components R, G, B (e.g., RGB, GBR, BGR, YUV) will be acceptable as far as the order of Quantization tables and Huffman tables match the order of image components.

7:6 **Reserved**

Exists If:	//Decoder Only
Format:	MBZ

7:6 **Reserved**

Project:	CHV, BSW
Exists If:	//Encoder Only
Format:	MBZ

5:4 **Rotation**

Exists If:	//Decoder Only

Value	Name	Description
00b		no rotation
01b		rotate clockwise 90 degree
10b		rotate counter-clockwise 90 degree (same as rotating 270 degree clockwise)

MFX_JPEG_PIC_STATE

11b		rotate 180 degree (NOT the same as flipped on the x-axis)

Programming Notes	Project
Rotation can be set to 01b, 10b, or 11b when OutputFormatYUV is set to 0000b. For other OutputFormatYUV, Rotation is not allowed.	CHV, BSW

5:3 **Reserved**

Project:	CHV, BSW
Exists If:	//Encoder Only
Format:	MBZ

3 **Reserved**

Exists If:	//Decoder Only
Format:	MBZ

2:0 **Input Format YUV**

Exists If:	//Decoder Only
Format:	U3

Value	Name	Description
0	**[Default]**	YUV400 (grayscale image)
1		YUV420
2		YUV422H_2Y (Horizontally chroma 2:1 subsampled) - horizontal 2 Y-block, 1U and 1V
3		YUV444
4		YUV411
5		YUV422V_2Y (Vertically chroma 2:1 subsampled) - vertical 2 Y-blocks, 1U and 1V
6		YUV422H_4Y - 2x2 Y-blocks, vertical 2U and 2V
7		YUV422V_4Y - 2x2 Y-blocks, horizontal 2U and 2V

2:0 **Output MCU Structure**

Project:	CHV, BSW
Exists If:	//Encoder Only

Output MCU Structure(**OutputMcuStructure**) should be set accordingly for each Input Surface Format YUV(**InputSurfaceFormatYUV**):

- If **InputSurfaceFormatYUV** is set to NV12, **OutputMCUStructure** is set to YUV420.
- If **InputSurfaceFormatYUV** is set to UYVY or YUY2, **OutputMCUStructure** is set to YUV422H_2Y.
- If **InputSurfaceFormatYUV** is set to Y8, **OutputMCuStructure** is set to YUV400.
- If **InputSurfaceFormatYUV** is set to RGB (or GBR, BGR, YUV), **OutputMCuStructure** is set to RGB.

MFX_JPEG_PIC_STATE

- If **InputSurfaceFormatYUV** is set to RGB, the order of encoded blocks in MCU will be same as the order of input image components. If the order of input image components is RGB (or GBR, BGR, YUV), then the order of blocks will be RGB (or GBR, BGR, YUV respectively).

Value	Name	Description
0	YUV400	Grayscale Image
1	YUV420	Both horizontally and vertically chroma 2:1 subsampled
2	YUV422H_2Y	Horizontally chroma 2:1 subsampled - horizontal 2 Y-blocks, 1 U and 1 V block
3	RGB	RGB or YUV444: No subsample
4		
5		
6		
7		

2 | 31:30 | Reserved

Exists If:	//Decoder Only
Format:	MBZ

31:29 | Reserved

Project:	CHV, BSW
Exists If:	//Encoder Only
Format:	MBZ

29 | Reserved

Exists If:	//Decoder Only
Format:	MBZ

28:16 | Frame Height In Blocks Minus 1

Exists If:	//Decoder Only
Format:	U13-1

Description
(The number of blocks in height) - 1. This value is calculated using the number of lines Y and vertical sampling factor of the first component V_1 in Frame header. See the note following this table. For interleaved components, (((Y + (V_1*8 -1)) / (V_1*8)) * V_1) - 1, where "/" is integer division. For non-interleaved components, ((Y + 7) / 8) - 1.

Workaround: For interleaved components, when **Input Format YUV** is set to **YUV422H_2Y**, **OutputFormatYUV** is set to **NV12**,
If ((((((Y + (V_1*8 -1)) / (V_1*8)) * V_1) - 1)% 2) == 0),
then **Frame Height In Blocks Minus 1** = ((Y + (V1*8 -1)) / (V1*8)) * V1
else

MFX_JPEG_PIC_STATE

		then **Frame Height In Blocks Minus 1** = (((Y + (V1*8 -1)) / (V1*8)) * V1) - 1
	28:16	**Frame Height In Blks Minus 1** Project: CHV, BSW Exists If: //Encoder Only Format: U13-1 (The number of blocks in height) - 1. This value is calculated using the number of lines Y and vertical sampling factor of the first component V1 in Frame header. See the note following this table. For interleaved components: (((Y + (V1*8 -1)) / (V1*8)) * V1) - 1 For non-interleaved components: ((Y + 7) / 8) - 1
	15:13	**Reserved** Exists If: //Decoder Only Format: MBZ
	15:13	**RoundingQuant** Project: CHV, BSW Exists If: //Encoder Only Rounding value applied to quantization output

Value	Name	Description
000b	**[Default]**	1/2
001b		(1/2 - 1/128)
010b		(1/2 + 1/128)
011b		(1/2 - 1/64)
100b		(1/2 + 1/64)
101b		(1/2 - 1/32)
110b		(1/2 - 1/16)
111b		(1/2 - 1/8)

	12:0	**Frame Width In Blocks Minus 1** Exists If: //Decoder Only Format: U13-1 (The number of blocks in width) - 1. This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H_1 in Frame header. See the note following this table. For interleaved components, (((X + (H_1 *8 -1)) / (H_1 *8)) * H_1) - 1. For non-interleaved components, ((X + 7) / 8) - 1.
	12:0	**Frame Width In Blks Minus 1** Project: CHV, BSW Exists If: //Encoder Only Format: U13-1

MFX_JPEG_PIC_STATE		
		(The number of blocks in width) - 1. This value is calculated using the number of samples per line X and horizontal sampling factor of the first component H1 in Frame header. See the note following this table.
		For interleaved components: (((X + (H1 *8 -1)) / (H1 *8)) * H1) - 1 For non-interleaved components: ((X + 7) / 8) - 1

MFX_MPEG2_PIC_STATE

MFX_MPEG2_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands. For MPEG-2 the encoder is called per slice-group, however the picture state is called per picture.Notice that a slice-group is a group of consecutive slices that no non-trivial slice headers are inserted in between.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MPEG2_PIC_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 3h MPEG2_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1)= 00Bh, used for normal decode and encode mode000h, a special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware. Format: =n Total Length - 2
1	31:28	**f_code[1][1].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	27:24	**f_code[1][0].** Used for backward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details

MFX_MPEG2_PIC_STATE

	23:20	**f_code[0][1]** Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	19:16	**f_code[0][0]** Used for forward motion vector prediction. See ISO/IEC 13818-2 7.6.3.1 for details
	15:14	**Intra DC Precision** Project: All Format: U2 See ISO/IEC 13818-2 6.3.10 for details.
	13:12	**Picture Structure** This field specifies whether the picture is encoded in the form of a frame picture or one field (top or bottom) picture. See ISO/IEC 13818-2 6.3.10 for details.Format = MPEG_PICTURE_STRUCTURE00 = Reserved01 = MPEG_TOP_FIELD10 = MPEG_BOTTOM_FIELD11 = MPEG_FRAME
	11	**TFF (Top Field First)** When two fields are stored in a picture, this bit indicates if the top field is the first field.For a frame P picture, the value 1 indicates that the top field of the reconstructed frame is the first field output by the decoding process, the same as defined in ISO/IEC 13818-2 6.3.10. Particularly, it is used by the hardware to calculate derivative motion vectors from the dual-prime motion vectors.For a field P picture, hardware uses this bit together with the Picture Structure to determine if the current picture is the Second Field. In this case, the definition of this bit differs from ISO/IEC 13818-2 6.3.10 - software must derive the value for this bit according to the following relation:Picture Structure = top fieldPicture Structure = bottom fieldSecond Field = 0TFF = 1TFF = 0Second Field = 1TFF = 0TFF = 1
	10	**Frame Prediction Frame DCT** This field provides constraints on the DCT type and prediction type. It affects the syntax of the bitstream.
	9	**Concealment Motion Vector Flag** This field indicates if the concealment motion vectors are coded in intra macroblocks. It affects the syntax of the bitstream.
	8	**Quantizer Scale Type** Format: MPEG_Q_SCALE_TYPE This field specifies the quantizer scaling type. Value / Name / Description 0h / / MPEG_QSCALE_LINEAR 1h / / D MPEG_QSCALE_NONLINEAR esc
	7	**Intra VLC Format** This field is used by VLD
	6	**Scan Order** Format: MPEG_INVERSESCAN_TYPE

MFX_MPEG2_PIC_STATE

This field specifies the Inverse Scan method for the DCT-domain coefficients in the blocks of the current picture.

Value	Name	Description
0h		MPEG_ZIGZAG_SCAN
1h		MPEG_ALTERNATE_VERTICAL_SCAN

Reserved (5:0)

I Slice Concealment Mode (2, 31)

Project:	CHV, BSW
Exists If:	//Decoder

This field controls how MPEG decoder handles MB concealment in I Slice

Value	Name	Description
0h	Intra Concealment	Using Coefficient values to handle MB concealment
1h	Inter Concealment	Using Motion Vectors to handle MB concealment

Programming Notes
If this field is set to "1", driver must provide a valid forward reference picture (both top and bottom Field must be valid)

Reserved (30)

Project:	CHV, BSW
Format:	MBZ

P/B Slice Concealment Mode (29:28)

Project:	CHV, BSW
Exists If:	//Decoder

This field controls how MPEG decoder handles MB concealment in P/B Slice.

Value	Name	Description
00b	INTER	If left MB is NOT Intra MB type (including skipMB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use forward reference (same polarity for field pic) with MV final values set to 0.
01b	LEFT	If left MB is NOT Intra MB type (including skipMB), use left MB inter prediction mode [frame/field or forward/backward/bi] and MV final values as concealment. Otherwise (left MB is Intra MB), use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data)
10b	ZERO	Always use forward reference (same polarity for field pic) with MV final values set to 0 (Macroblock is concealed as INTER coded)
11b	INTRA	Use left MB dct_dc_pred[cc] values for concealment (Macroblock is concealed as INTRA MB and dct_dc_pred[cc] are DC predictor for Luma, Cr, Cb data

MFX_MPEG2_PIC_STATE

27 **Reserved**

Project:	CHV, BSW
Format:	MBZ

26:25 **P/B Slice Predicted BiDir Motion Type Override - Bi-direction MV Type Override**

Project:	CHV, BSW
Exists If:	//Decoder

This field is only applicable if the Concealment Motion Type is predicted to be Bi-directional. (It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB is a bi-directional MB).

Value	Name	Description
0h	BID	Keep Bi-direction Prediction
1h	RESERVED	
2h	FWD	Only use Forward Prediction (Backward MV is forced to invalid
3h	BWD	Only use Backward Prediction (Forward MV is forced to invalid)

24 **P/B Slice Predicted Motion Vector Override Final MV value Override**

Project:	CHV, BSW
Exists If:	//Decoder

This field is only applicable if the Concealment Motion Vectors are non-zero. It is only possible if "P/B Slice Concealment Mode" is set to "00" or "01" and left MB has non-zero motion vectors).

Value	Name	Description
0h	Predicted	Motion Vectors use predicted values
1h	ZERO	Motion Vectors force to 0

23:15 **Reserved**

Format:	MBZ

14 **LoadSlicePointerFlag - LoadBitStreamPointerPerSlice**

Exists If:	//Encoder

To support multiple slice picture and additional header/data insertion before and after an encoded slice.When this field is set to 0, bitstream pointer is only loaded once for the first slice of a frame. For subsequent slices in the frame, bitstream data are stitched together to form a single output data stream.When this field is set to 1, bitstream pointer is loaded for each slice of a frame. Basically bitstream data for different slices of a frame will be written to different memory locations.

Value	Name	Description
0h		Load BitStream Pointer only once for the first slice of a frame
1h		Load/reload BitStream Pointer only once for the each slice, reload the start location of the bitstream buffer from the Indirect PAK-BSE Object Data Start Address field

13 **Reserved**

MFX_MPEG2_PIC_STATE

	12	**Reserved** Format: MBZ
	11	**Reserved** Format: MBZ
	10:9	**Picture Coding Type** Format: MPEG_PICTURE_CODING_TYPE This field identifies whether the picture is an intra-coded picture (I), predictive-coded picture (P) or bi-directionally predictive-coded picture (B). See ISO/IEC 13818-2 6.3.9 for details. Value / Name: 00b Reserved 01b MPEG_I_PICTURE 10b 10 = MPEG_P_PICTURE 11b MPEG_B_PICTURE
	8:2	**Reserved** Format: MBZ
	1	**MismatchControlDisabled** These 2 bits flag disables mismatch control of the inverse transformation for some specific cases during reference reconstruction. Value / Name / Description: 00b — Mismatch control applies to all MBs 01b — Disable mismatch control to all intra MBs whose all AC-coefficients are zero. 10b — Disable mismatch control to all MBs whose all AC-coefficients are zero. 11b — Disable mismatch control to all MBs.
	0	**Disable Mismatch** To disable MPEG2 IDCT fixed point arithmetic correction
3	31	**Slice Concealment Disable Bit** Project: CHV, BSW Exists If: //Decode If VINunit detects the next slice starting position is either out-of-bound or smaller than or equal to the current slice starting position, VIN will set the current slice to be 1 MB and force VMDunit to do slice concealment on the next slice. This bit will disable this feature and the MB data from the next slice will be decoded from bitstream. Value / Name / Description: 0h — Enable **[Default]** — VIN will force next slice to be concealment if detects slice boundary error 1h — Disable — VIN will not force next slice to be in concealment

MFX_MPEG2_PIC_STATE

		Programming Notes Driver has an option to detect the scenario given in description (above) and remove the second (out-of-order) slice. In this case, hardware will decode the first slice in completion and do concealment till the third slice. It should yield a picture with better quality this way.
	30:29	**Reserved** Format: MBZ
	28:24	**Reserved**
	23:16	**FrameHeightInMBsMinus1[7:0] (Picture Height in Macroblocks)** Format: U8
	15:8	**Reserved** Format: MBZ for future supporting width > 4K
	7:0	**FrameWidthInMBsMinus1[7:0] (Picture Width in Macroblocks)** Project: All Format: U8
4	31:16	**MinFrameWSize** Project: All Format: U16 - Minimum Frame Size [15:0] (16-bit) (Encoder Only)Mininum Frame Size is specified to compensate for intel Rate ControlCurrently zero fill (no need to perform emulation byte insertion) is done only to the end of the CABAC_ZERO_WORD insertion (if any) at the last slice of a picture. Intel encoder parameter, not part of DXVA. The caller should always make sure that the value, represented by Mininum Frame Size, is always less than maximum frame size FrameBitRateMax (DWORD 10 bits 29:16). This field is reserved in Decode mode.
		Value / **Name** / **Description** [0,0003FFFFh] / / The programmable range when MinFrameWSizeUnits is 00. [0,000FFFFFh] / / The Programmable range when MinFrameWSizeUnits is 01. [0,03FFFFFFh] / / The Programmable range when MinFrameWSizeUnits is 10. [0, FFFFFFFFh] / / The Programmable range when MinFrameWSizeUnits is 11. 0h / **[Default]** /
	15	**Reserved** Project: All Format: MBZ
	14:12	**RoundInterAC,** rounding precision for non-Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	11	**Reserved**

MFX_MPEG2_PIC_STATE

	10:8	**RoundIntraAC** Project: All Format: U3 rounding precision for Intra AC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	7	**Reserved** Format: MBZ
	6:4	**RoundInterDC** rounding Precision for non-Intra-DC000: +1/16001: +2/16010: +3/16011: +4/16100: +5/16101: +6/16110: +7/16111: +8/16
	3	**Reserved** Format: MBZ
	2:1	**RoundIntraDC** rounding Precision for Intra-DC00: +1/801: +2/810: +3/811: +4/8
	0	**Reserved**
5	31:17	**Reserved** (for future Mask bits)
	16	**FrameSizeControlMask** Frame size conformance maskThis field is used when MacroblockStatEnable is set to 1.

Value	Name	Description
0h		Do not change Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control
1h		Replace Slice Quantization Parameter values in MFC_MPEG2_SLICEGROUP_STATE with suggested slice QP value for frame level Rate control values in MFC_IMAGE_STATUS control register.

	15:13	**Reserved**
	12	**InterMBForceCBPZeroControlMask** Format: U1 Inter MB Force CBP ZERO mask.

Value	Name	Description	Project
[0, FFFFFFFFh]			
0h		No effect	All
1h		Zero out all A/C coefficients for the inter MB violating Inter Confirmance	All

	11:10	**MinFrameWSizeUnits** This field is the Minimum Frame Size Units

MFX_MPEG2_PIC_STATE

Value	Name	Description
00b	compatibility mode	Minimum Frame Size is in old mode (words, 2bytes)
01b	16 byte	Minimum Frame Size is in 16bytes
10b	4Kb	Minimum Frame Size is in 4Kbytes
11b	16Kb	Minimum Frame Size is in 16Kbytes

9 **MBRateControlMask**
This field is ignored when MacroblockStatEnable is disabled or MB level Rate control flag for the current MB is disable in Macroblock Status Buffer.

Value	Name	Description
0h		Do not change QP values of inter macroblock with suggested QP values in Macroblock Status Buffer
1h		Apply RC QP delta for all macroblock

8 **Reserved**

7 **Reserved**

6:4 **Reserved**

3 **FrameBitRateMinReportMask**
This is a mask bit controlling if the condition of frame level bit count is less than FrameBitRateMin.

Value	Name	Description	Project
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.	All
1h	Enable	set bit0 and bit 1of MFC_IMAGE_STATUS control register if the total frame level bit counter is less than or equal to Frame Bit rate Minimum limit.	All

2 **FrameBitRateMaxReportMask**
This is a mask bit controlling if the condition of frame level bit count exceeds FrameBitRateMax.

Value	Name	Description	Project
0h	Disable	Do not update bit0 of MFC_IMAGE_STATUS control register.	All
1h	Enable	set bit0 and bit 1 of MFC_IMAGE_STATUS control register if the total frame level bit counter is greater than or equal to Frame Bit rate Maximum limit.	All

1 **InterMBMaxSizeReportMask**
This is a mask bit controlling if the condition of any inter MB in the frame exceeds InterMBMaxSize.

Value	Name	Description
0h		Do not update bit0 of MFC_IMAGE_STATUS control register.
1h		set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Inter MB Conformance Max size limit.

MFX_MPEG2_PIC_STATE

	0	**IntraMBMaxSizeReportMask** This is a mask bit controlling if the condition of any intra MB in the frame exceeds IntraMBMaxSize. **Value** / **Name** / **Description** 0h / / Do not update bit0 of MFC_IMAGE_STATUS control register. 1h / / set bit0 of MFC_IMAGE_STATUS control register if the total bit counter for the current MB is greater than the Intra MB Conformance Max size limit.
6 [ExistsIf]Encode Only	31:28	**Reserved** Format: MBZ
	27:16	**InterMBMaxSize** Default Value: FFFh This field, Inter MB Conformance Max size limit,indicates the allowed max bit count size for Inter MB
	15:12	**Reserved** Format: MBZ
	11:0	**IntraMBMaxSize** Default Value: FFFh This field, Intra MB Conformance Max size limit,indicates the allowed max bit count size for Intra MB
7	31:1	**Reserved**
	0	**VSL top MB Trans8x8flag** Project: CHV, BSW Exists If: //Encode Only **Value** / **Name** / **Description** 0 / Disable / VSL will only fetch the current MB data. 1 / Enable / When this bit is set VSL will make extra fetch to memory to fetch the MB data for top MB.
8 [ExistsIf]Encode Only	31:24	**SliceDeltaQPMax[3]** Format: S7 This field is the Slice level delta QP for total bit-count above FrameBitRateMax - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta»3). Range: [-30,30]

MFX_MPEG2_PIC_STATE

		Value / **Name** 0h / Disable 1h / Enable
	23:16	**SliceDeltaQPMax[2]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/8 and below 1/ 4 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and ¼ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»3), (FrameBitRateMax+ FrameBitRateMaxDelta»2).
	15:8	**SliceDeltaQPMax[1]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above1/ 4 and below 1/2 This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between ¼ and ½ of FrameBitRateMaxDelta above FrameBitRateMax, i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»2), (FrameBitRateMax+ FrameBitRateMaxDelta»1).
	7:0	**SliceDeltaQPMax[0]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count above FrameBitRateMax - above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta , i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta»1), infinite).
9 [ExistsIf]Encode Only	31:24	**SliceDeltaQPMin[3]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for total bit-count below FrameBitRateMin - first 1/8 regionThis field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin-FrameBitRateMinDelta»3), FrameBitRateMin).

MFX_MPEG2_PIC_STATE

	23:16	**SliceDeltaQPMin[2]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count below FrameBitRateMin - below 1/ 8 and above 1/ 4This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»2), (FrameBitRateMin- FrameBitRateMinDelta»3)).
	15:8	**SliceDeltaQPMin[1]** Format: S7 Range: [-30,30] This field is the Slice level delta QP for bit-count below FrameBitRateMin- below 1/4 and above 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin, i.e., in the range of [(FrameBitRateMin- FrameBitRateMinDelta»1), (FrameBitRateMin- FrameBitRateMinDelta»2)).
	7:0	**SliceDeltaQPMin[0]** Format: S7 Range: [-30,30] This field is the Slice Level Delta QP for bit-count below FrameBitRateMin - below 1/ 2This field is used to calculate the suggested slice QP into the MFC_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta , i.e., in the range of [0, (FrameBitRateMin- FrameBitRateMinDelta»1).
10 [ExistsIf]Encode Only	31	**FrameBitrateMaxUnit** This field is the Frame Bitrate Maximum Limit Units.

Value	Name	Description	Project
0h	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMaxUnitMode is 1 and in units of 128 Bytes if FrameBitrateMaxUnitMode is 0	All
1h	Kilobyte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0	All

MFX_MPEG2_PIC_STATE

	30	**FrameBitrateMaxUnitMode** BitFiel This field is the Frame Bitrate Maximum Limit Units.dDesc

Value	Name	Description	Project
0h	Compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)	All
1h	New mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)	All

	29:16	**FrameBitRateMax** This field is the Frame Bitrate Maximum Limit. This field along with FrameBitrateMaxUnit determines maximum allowed bits in a frame before multi-pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count exceeds this value. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28 and 29 should be 0.

Value	Name	Description
0-512KB		The programmable range 0-512KB when FrameBitrateMaxUnit is 0.
0-8190KB		The programmable range 0-8190KB when FrameBitrateMaxUnit is 1.

	15	**FrameBitrateMinUnit** This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description	Project
0h	Byte	FrameBitRateMax is in units of 32 Bytes when FrameBitrateMinUnitMode is 1 and in units of 128 Bytes if FrameBitrateMinUnitMode is 0	All
1h	KiloByte	FrameBitRateMax is in units of 4KBytes Bytes when FrameBitrateMaxUnitMode is 1 and in units of 16KBytes if FrameBitrateMaxUnitMode is 0	All

	14	**FrameBitrateMinUnitMode** This field is the Frame Bitrate Minimum Limit Units.ValueNameDescriptionProject

Value	Name	Description	Project
0h	compatibility mode	FrameBitRateMaxUnit is in old mode (128b/16Kb)	All
1h	New Mode	FrameBitRateMaxUnit is in new mode (32byte/4Kb)	All

	13:0	**FrameBitRateMin** This field is the Frame Bitrate Minimum Limit ()This field along with FrameBitrateMinUnit determines minimum allowed bits in a Frame before Multi-Pass gets triggered (when enabled). In other words, multi-pass is triggered when the actual frame byte count is less than this value. When FrameBitrateMinUnitMode is 0 (compatibility mode) bits 0:11 should be used, bits 12 and 13 should be 0. Range: The programmable range 0-512KB When FrameBitrateMinUnit is in 0. Programmable range is 0-8190 KB when FrameBitrateMinUnit is in 1
11	31	**Reserved**

MFX_MPEG2_PIC_STATE

[ExistsIf]Encode Only

30:16 **FrameBitRateMaxDelta**

Default Value:	0h
Project:	All
Access:	None
Format:	U15

This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. The programmable range is either 0- 512KB or 4MBB in FrameBitrateMaxUnit of 128 Bytes or 16KB respectively.
This field is used to select the slice delta QP when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. When FrameBitrateMaxUnitMode is 0(compatibility mode) bits 16:27 should be used, bits 28, 29 and 30 should be 0.

15 **Reserved**

Project:	All
Format:	MBZ

14:0 **FrameBitRateMinDelta**

This field is used to select the slice delta QP when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. When FrameBitrateMinUnitMode is 0(compatibility mode) bits 0:11 should be used, bits 12, 13 and 14 should be 0.Note: HW requires the following condition FrameBitRateMinDelta <= 2*FrameBitRateMinMust be true, otherwise it may cause unpredicted behavior.

Value	Name	Description
0-1024KB		The programmable range 0-1024KB When FrameBitrateMinUnit is in 32Bytes.
0-16380KB		Programmable range is 0-16380KB when FrameBitrateMinUnit is in 4Kbytes.

12

31:21 **Reserved**

Format:	MBZ

20 **VMD Error Logic**

Project:	CHV, BSW

Value	Name	Description
0	Disable **[Default]**	
1	Enable	Error Handling

19 **Reserved**

Format:	MBZ

18 **VAD Error Logic**

Project:	CHV, BSW

MFX_MPEG2_PIC_STATE

Value	Name	Description
0	Enable **[Default]**	Error reporting ON in case of premature Slice done
1	Disable	CABAC Engine will auto decode the bitstream in case of premature slice done.

17	**Reserved**
	Project: CHV, BSW
16	**Reserved**
	Project: CHV, BSW
15:0	**Reserved**
	Format: MBZ

MFX_PAK_INSERT_OBJECT

MFX_PAK_INSERT_OBJECT

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

Description

The MFX_PAK_INSERT_OBJECT command is the first primitive command for the AVC and MPEG2 Encoding Pipeline. The MFX_PAK_INSERT_OBJECT command is the first primitive command for the AVC, MPEG2, JPEG, and VP8 Encoding Pipeline.

This command is issued to setup the control and parameters of inserting a chunk of compressed/encoded bits into the current bitstream output buffer starting at the specified bit locationto perform the actual insertion by transferring the command inline data to the output buffer max, 32 bits at a time.
It is a variable length command as the data to be inserted are presented as inline data of this command. It is a multiple of 32-bit (1 DW), as the data bus to the bitstream buffer is 32-bit wide.
Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid H.264 bitstream.
Internally, MFX hardware will keep track of the very last two bytes' (the very last byte can be a partial byte) values of the previous insertion. It is required that the next Insertion Object Command or the next PAK Object Command to perform the start code emulation sequence check and prevention 0x03 byte insertion with this end condition of the previous insertion.
Hardware will keep track of an output bitstream buffer current byte position and the associated next bit insertion position index. Data to be inserted can be a valid H.264 NAL units or a partial NAL unit. Certain NAL unit has a minimum byte size requirement. As such the hardware will optionally (enabled by STATE Command) determines the number of CABAC_ZERO_WORD to be inserted to the end of the current NAL, based on the minimum byte size of a NAL and the actual bin count of the encoded Slice. Since prior to the CABAC_ZERO_WORD insertion, the RBSP or EBSP is already byte-aligned, so each CABAC_ZERO_WORD insertion is actually a 3-byte sequence 0x00 00 03. The inline data may have already been processed for start code emulation byte insertion, except the possibility of the last 2 bytes plus the very last partial byte (if any). Hence, when hardware performing the concatenation of multiple consecutive insertion commands, or concatenation of an insertion command and a PAK object command, it must check and perform the necessary start code emulation byte insert at the junction.The inline data is required to be byte aligned on the left (first transmitted bit order) and may or may not be byte aligned on the right (last transmitted bits).
The command will specify the bit offset of the last valid DW.Each insertion state command defines a chunk of bits (compressed data) to be inserted at a specific location of the output compressed bitstream in the output buffer.Depend on CABAC or CAVLC encoding mode (from Slice State), PAK Object Command is always ended in byte aligned output bitstream except for CABAC header insertion which is bit aligned. In the aligned cases, PAK will perform 0 filling in CAVLC mode, and 1 filling in CABAC mode.
Insertion data can include:any encoded syntax elements bit data before the encoded Slice Data (PAK Object Command) of the current SliceSPS NALPPS NALSEI NALOther Non-Slice NALLeading_Zero_8_bits (as many bytes as there is)Start Code PrefixNAL Header ByteSlice HeaderAny encoded syntax elements bit data after the encoded Slice Data (PAK Object Command) of the current Slice and prior to the next encoded Slice Data of the next Slice or prior to the end of the bistream, whichever comes firstCabac_Zero_Word or Trailing_Zero_8bits (as many bytes as there is).
Anything listed above before a Slice DataContext switch interrupt is not supported by this command.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="8">0</td><td>31:29</td><td>Command Type<table><tr><td>Default Value:</td><td>3h PARALLEL_VIDEO_PIPE</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>28:27</td><td>Pipeline<table><tr><td>Default Value:</td><td>2h MFX_PAK_INSERT_OBJECT</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>26:24</td><td>Media Command Opcode<table><tr><td>Default Value:</td><td>0h MFX_COMMON</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>23:21</td><td>SubOpcode A<table><tr><td>Default Value:</td><td>2h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>20:16</td><td>SubOpcode B<table><tr><td>Default Value:</td><td>8h</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td>15:12</td><td>Reserved</td></tr>
<tr><td>11:0</td><td>DWord Length<table><tr><td>Default Value:</td><td>0h Excludes DWord (0,1) = Variable Length in DW</td></tr><tr><td>Format:</td><td>=n Total Length - 2</td></tr></table></td></tr>
<tr><td rowspan="3">1</td><td>31:18</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td>17:16</td><td>DataByteOffset - SrcDataStartingByteOffset[1:0]
Source Data Starting Byte Position within the very first inline DW.<table><tr><th>Programming Notes</th></tr><tr><td>Must be set to 0 for JPEG encoder</td></tr></table></td></tr>
<tr><td>15</td><td>HeaderLengthExcludeFrmSize
In case this flag is on, bits are NOT accumulated during current access unit coding neither for Cabac Zero Word insertion bits counting or for output in MMIO register MFC_BITSTREAM_BYTECOUNT_FRAME_NO_HEADER. When using HeaderLenghtExcludeFrmSize for header insertion, the software needs to make sure that data comes already with inserted start code emulation bytes. SW shouldn't set EmulationFlag bit (Bit 3 of DWORD1 of MFX_PAK_INSERT_OBJECT).<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>1</td><td>NO_ACCUMULATION</td><td>Bits during current call are not accumulated</td></tr><tr><td>0</td><td>ACCUMULATE</td><td>All bits accumulated</td></tr></table><table><tr><th>Programming Notes</th><th>Project</th></tr><tr><td>Must be set to 0 for JPEG encoder</td><td>CHV, BSW</td></tr></table></td></tr>
</table>

MFX_PAK_INSERT_OBJECT

14 **Slice Header Indicator**

This bit indicates if the insert object is a slice header. In the VDEnc mode, PAK only gets this command at the beginning of the frame for slice position X=0, Y=0. It internally generates the header that needs to be inserted per slice. For VDEnc mode, this bit should always be set.

Value	Name	Description
1	SLICE_HEADER	Insertion Object is a Slice Header. The command is stored internally by HW and is used for inserting slice headers.
0	LEGACY	Legacy Insertion Object command. The PAK Insertion Object command is not stored in HW.

Programming Notes
In VDENC mode, we support only Slice layer without partitioning RBSP syntax. The payload for PAK_INS_OBJ should contain only start code for Slice header followed by NAL_type and slice header (slice_header() in AVC spec). The payload for PAK_INS_OBJ shouldn't contain CABAC Byte alignment bits. HW adds these alignment bits which are part of slice_data. Example PAK_INS_OBJ payload : 00 00 01 <NAL_type> <slice_header_Byte0> .....<slice_header_Byte LAST> Any zero_bytes that are added before slice header can be inserted by any preceding general PAK_INS_OBJ.

13:8 **DataBitsInLastDW - SrCDataEndingBitInclusion[5:0]**

Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first.For example, SrCDataEndingBitInclusion = 9, bit 7:0 and bit 15 are included as valid header data.

Value	Name
[1,32]	

7:4 **SkipEmulByteCnt - Skip Emulation Byte Count**

Skip emulation check for number of starting bytesIt can be programmed from 0 to 15 bytes.For example, to skip the start code that has already prefixed in the bitstream.

Programming Notes
Must be set to 0 for JPEG encoder

3 **EmulationFlag - EmulationByteBitsInsertEnable**

Value	Name	Description
0	NONE	No emulation
1	EMULATE	Instruct the hardware to perform Start Code Prefix (0x 00 00 01/02/03/00) Search and Prevention Byte (0x 03) insertion on the insertion data of this command. It is required that hardware will handle a start code prefix crossing the boundary between insertion commands, or an insertion command followed by a PAK Object command.

Programming Notes
Must be set to 0 for JPEG encoder

MFX_PAK_INSERT_OBJECT

	2	**LastHeaderFlag - LastSrcHeaderDataInsertCommandFlag** To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series.In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.In CAVLC, hardware ignores this bit
	1	**EndOfSliceFlag - LastDstDataInsertCommandFlag** No more insertion command and no more PAK-OBJECT command follows.Flush data out to memory
	0	**BitstreamStartReset - ResetBitStreamStartingPos** Value / Name / Description: 1 / RESET / Reset the bitstream buffer insertion position to the bitstream buffer starting position. 0 / INSERT / Insert the current command inline data starting at the current bitstream buffer insertion position **Programming Notes** Must be set to 1 for JPEG encoder
2..n	31:0	**Insert Data PayLoad** Actual Data to be inserted to the output bitstream buffer.

MFX_PIPE_BUF_ADDR_STATE

MFX_PIPE_BUF_ADDR_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This state command provides the memory base addresses for all row stores, StreamOut buffer and reconstructed picture output buffers required by the MFD or MFC Engine (that are in addition to the row stores of the Bit Stream Decoding/Encoding Unit (BSD/BSE) and the reference picture buffers). This is a picture level state command and is common among all codec standards and for both encoder and decoder operating modes. However, some fields may only applicable to a specific codec standard. All Pixel Surfaces (original, reference frame and reconstructed frame) in the Encoder are programmed with the same surface state (NV12 and TileY format), except each has its own frame buffer base address. In the tile format, there is no need to provide buffer offset for each slice; since from each MB address, the hardware can calculated the corresponding memory location within the frame buffer directly.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_PIPE_BUF_ADDR_STATE Format: OpCode
	26:24	**Common Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Fixed Length

Value	Name	Description	Project
3Bh	DWORD_COUNT_n **[Default]**	Excludes DWord (0,1)	CHV, BSW

MFX_PIPE_BUF_ADDR_STATE

1 — 31:6 — Pre Deblocking Destination Address

Format:	GraphicsAddress[31:6]

Specifies the 4K byte aligned frame buffer address for outputting the non-filtered reconstructed YUV picture (i.e. output of final adder in each codec standard, and prior to the deblocking filter unit). This field is ignored if PreDeblockOutEnable is set to 0 (disable).

1 — 5:0 — Reserved

Project:	CHV, BSW

2 — 31:16 — Reserved

Format:	MBZ

2 — 15:0 — Pre Deblocking Destination Address High

Project:	CHV, BSW
Format:	GraphicsAddress[47:32]

This field is for the upper range of Pre-Deblocking Destination Address. This field is ignored if **PreDeblockOutEnable** is set to 0 (disable).

3 — 31:15 — Reserved

Format:	MBZ

3 — 14:13 — Reserved

3 — 12:11 — Reserved

Format:	MBZ

3 — 10:9 — Reserved

Project:	CHV, BSW
Format:	MBZ

3 — 8:7 — Pre Deblocking - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

3 — 6:5 — Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Pre Deblocking Destination Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

MFX_PIPE_BUF_ADDR_STATE

		Value / **Name** / **Description** 00b / Use Cacheability Controls from page table / 01b / UC / Uncacheable - non-cacheable 10b / WT / Writethrough 11b / WB / Writeback
	4:3	**Target Cache (TC) Pre Deblocking Destination Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching. 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) Pre Deblocking Destination Address** Project: CHV, BSW This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11b: Good chance of generating hits 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
4	31:6	**Post Deblocking Destination Address** Format: GraphicsAddress[31:6] Specifies the 4K byte aligned frame buffer address for outputting the post-loop filtered reconstructed YUV picture (i.e. output of the deblocking filter unit)This field is ignored if PostDeblockOutEnable is set to 0 (disable).
	5:0	**Reserved** Project: CHV, BSW
5	31:16	**Reserved** Format: MBZ
	15:0	**Post Deblocking Destination Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]

MFX_PIPE_BUF_ADDR_STATE

This field is for the upper range of Post-Deblocking Destination Address. This field is ignored if **PostDeblockOutEnable** is set to 0 (disable).

6 — 31:15 Reserved

Format:	MBZ

14:13 Reserved

Project:	CHV, BSW
Format:	MBZ

12:11 Reserved

Format:	MBZ

10:9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 Post Deblocking - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Post Deblocking Destination Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 Target Cache (TC) for Post Deblocking Destination Address

Project:	CHV, BSW

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for

MFX_PIPE_BUF_ADDR_STATE

a memory access to be allocated in L3$ as well as LLC/eLLC
00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)*
01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*).
10b: LLC/eLLC Allowed.
11b: L3, LLC, eLLC Allowed.
Errata CHV:A-E (FIXED BY:G0 Stepping):
For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching.
For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops.
Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

Project:	CHV, BSW

1:0 **Age for QUADLRU (AGE) for Post Deblocking Destination Address**

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent good chance of generating hits
00b	Poor good chance of generating hits

7 31:6 **Original Uncompressed Picture Source Address**

Format:	GraphicsAddress[31:6]

Specifies the 64 byte aligned frame buffer address for fetching YUV pixel data from the original uncompressed input picture for encoding. This field is only valid in **encoding** mode.

5:0 **Reserved**

MFX_PIPE_BUF_ADDR_STATE

		Project: CHV, BSW Format: MBZ
8	31:16	**Reserved** Format: MBZ
	15:0	**Original Uncompressed Picture Source Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Original Uncompressed Picture Source Address. This field is valid for **encoding** mode only.
9	31:15	**Reserved** Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Format: MBZ
	8:7	**Original Uncompressed Picture Source - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Original Uncompressed Picture Source Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

MFX_PIPE_BUF_ADDR_STATE

	4:3	**Target Cache (TC) for Original Uncompressed Picture Source Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Original Uncompressed Picture Source Address** Project: CHV, BSW This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
10	31:6	**StreamOut Data Destination Base Address** Format: GraphicsAddress[31:6] Specifies the 64 byte aligned address for outputting the per-MB indirect data to memory when **StreamOutEnable** is set in the MFX_PIPE_MODE_SELECT command. For Decoder : This field is used for transcoding purpose. For Encoder : This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
11	31:16	**Reserved** Format: MBZ
	15:0	**StreamOut Data Destination Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Original Uncompressed Picture Source Address
12	31:15	**Reserved**

MFX_PIPE_BUF_ADDR_STATE

	Format:	MBZ

14:13 Reserved

Project:	CHV, BSW
Format:	MBZ

12:11 Reserved

Format:	MBZ

8:7 StreamOut Data Destination - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for StreamOut Data Destination Base Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 Target Cache (TC) for StreamOut Data Destination Base Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

Project:	CHV, BSW

1:0 Age for QUADLRU (AGE) for StreamOut Data Destination Base Address

MFX_PIPE_BUF_ADDR_STATE

		Project: CHV, BSW This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. Value / Name: 11b – Good chance of generating hits.; 10b – Next good chance of generating hits; 01b – Decent chance of generating hits; 00b – Poor chance of generating hits
13	31:6	**Intra Row Store Scratch Buffer Base Address** Format: GraphicsAddress[31:6] This field provides the base address of the scratch buffer (read/write) used by the AVC/VP8 IntraPrediction unit to store MB information of the previous row for processing of each macroblock in the current row. The Intra Row Store buffer must be 64-byte cacheline aligned.Hardware uses the horizontal address of the current macroblock to address the Intra Row Store. This field is ignored in MPEG2 and VC1 mode. Max 256 cachelines for 4K pixels (1 cacheline for either MBAFF or non-MBAFF)Intra Row Store Scratch Buffer - Arbitration Priority Control **Programming Notes** This is one of the four RowStore Scratch Buffers which can be programmed to use the internal Media Cache (total size 640 CacheLine). When Intra Row Store Scratch Buffer Cache Select is programmed to "1", this data will be stored inside MFX Media Internal Stroage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address to Media Cache. Driver needs to make sure the whole buffer fits into MFX Media Internal Storage. *(Notes: 1 cacheline per MB, and the buffer needs to have enough space for 1 MB row).*
	5:0	**Reserved**
14	31:16	**Reserved** Format: MBZ
	15:0	**Intra Row Store Scratch Buffer Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Intra RowStore/Scratch Buffer Base Address This field is ignored in MPEG2 and VC1 mode.
15	31:15	**Reserved** Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ

MFX_PIPE_BUF_ADDR_STATE

12 Intra Row Store Scratch Buffer Cache Select

Project:	CHV, BSW

This field controls if Intra Row Store is going to store inside Media Cache or to LLC.

Value	Name	Description
0		Buffer going to LLC.
1		Buffer going to Internal Media Storage

11 Reserved

Project:	CHV, BSW
Format:	MBZ

10:9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 Intra Row Store Scratch Buffer - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Intra Row Store Scratch Buffer Base Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 Reserved

2 Reserved

1:0 Age for QUADLRU (AGE) for Intra Row Store Scratch Buffer Base Address

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older

MFX_PIPE_BUF_ADDR_STATE

age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

16 | 31:6 | Deblocking Filter Row Store Scratch Base Address

Format:	GraphicsAddress[31:6]

Deblocking Filter Row Store is needed for:

- AVC and VC1 In-Loop Deblocking Filter
- VC1 Overlap-smoothing Filter
- AVC, VC1, and MPEG-2 Out-Of-Loop Deblocking Filter (Intel extension)

This field provides the 64 byte aligned base address of the scratch buffer (read and write) used by the deblocking filter unit to store MB information of the previous row for filtering of each macroblock in the current row. The Deblocking Filter Row Store buffer must be 64-byte cacheline aligned. Hardware uses the horizontal address of the current macroblock to address the Deblocking Filter Row Store. Max 6 cachelines for VC1 and MPEG2, and max 4 for AVC (for MBAFF, 2 for non-MBAFF)

Programming Notes

This is one of the four RowStore Scratch Buffers which can programmed to use the internal Media Cache (total size 640 CacheLine). When Deblocking Filter Row Store Scratch Buffer Cache Select is programmed to "1", this will be stored inside MFX Media Internal Storage. Driver then needs to program this Base Address between 0 to 639, indicating starting cachelines address location for this buffer. Driver needs to make sure the whole buffer fits into Media Internal Storage.
(Notes: 2 cachelines per MB for non-mbaff; 4 cahcelines per MB pair for mbaff, and the buffer needs to have enough space for 1 MB (pair) row).

5:0 | Reserved

17 | 31:16 | Reserved

15:0 | Deblocking Filter Row Store Scratch Base Address High

Project:	CHV, BSW
Format:	GraphicsAddress[47:32]

This field is for the upper range of Deblocking Filter Row Store Scratch Buffer Address.

18 | 31:15 | Reserved

14:13 | Reserved

Project:	CHV, BSW
Format:	MBZ

12 | Deblocking Filter Row Store Scratch Buffer Cache Select

MFX_PIPE_BUF_ADDR_STATE

		Project: CHV, BSW This field controls if Intra Row Store is going to store inside Media Internal Storage or to LLC. **Value** / **Name** / **Description** 0 — Buffer going to LLC 1 — Buffer going to Media Internal Storage
	11	**Reserved**
	10:9	**Reserved**
	8:7	**Deblocking Filter Row Store Scratch - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Deblocking Filter Row Store Scratch Base Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** / **Description** 00b — Use Cacheability Controls from page table 01b — UC — Uncacheable - non-cacheable 10b — WT — Writethrough 11b — WB — Writeback
	4:3	**Target Cache (TC) for Deblocking Filter Row Store Scratch Base Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching **Value** / **Name** 00b — eLLC Only - not snooped in GT 01b — LLC Only 10b — LLC/eLLC Allowed 11b — L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Deblocking Filter Row Store Scratch Base Address** This field allows the selection of AGE parameter for a given surface in LLC or eLLC.If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older

MFX_PIPE_BUF_ADDR_STATE

age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

19..50 63:48 Reserved

47:32 Reference Picture Address [n] High

Project:	CHV, BSW
Format:	Address[47:32]

This field is for the upper range of Reference Picture Addresses

31:6 Reference Picture Address [n]

Format:	Address[31:6]

Specifies the 64 byte aligned reference frame buffer addresses for the motion compensation operation in AVC/ /MPEG2. AVC can specify up to 16 YUV frame-based surfaces for both forward and backward references, i.e. L0+L1 total = 16 max. Any entry can be assigned to L0 or L1 or both lists.But VC1 and MPEG2, worst case, can use up to 2 YUV frame-based surfaces for both forward and backward references:

- P-MB : RefAddr[0] - temporal closest previous field of a reference frame (can be the current frame)
- RefAddr[1]- next temporal closest previous field of a reference frame (must be different from the current frame)

It is a variant (without the LongTermRefPic specification) of the RefFrameList[16] defined in AVC DXVA Spec. RefAddr[0-15] is indexed by frame_storeID »1. It is not a packed list, i.e. invalid entries can scatter among the list. All invalid addresses must be set to a valid address RefAddr[0] by the driver. The same applies to VC1 and MPEG2.

Programming Notes
AVC: Always specifies all 16 addresses even some of them are not needed as indicated by the max num of active reference pictures. This is done for preventing data corruption (error, fault condition, etc.) by having all the references being set to a legal location.

5:0 Reserved

Project:	CHV, BSW
Format:	MBZ

51 31:15 Reserved

Format:	MBZ

14:13 Reserved

Project:	CHV, BSW

MFX_PIPE_BUF_ADDR_STATE

	Format:	MBZ

12:9	**Reserved** Format: MBZ

8:7 Reference Picture - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Reference Picture Addresses

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Note: There is ONLY ONE LLC/eLLC Cacheability Control (LeLLCCC) for all 16 Reference Picture Addresses (RefAddr[0-15])**

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 Target Cache (TC) for Reference Picture Addresses

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching. **NOTE: There is ONLY ONE Target Cache (TC) for all 16 Reference Picture Addresses (RefAddr[0-15])**

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

Project:	CHV, BSW

1:0 Age for QUADLRU (AGE) for Reference Picture Addresses

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC.If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces

MFX_PIPE_BUF_ADDR_STATE

		are more likely to generate HITs, hence need to be replaced least often in caches. **NOTE: There is ONLY ONE Age for QUADLRU (AGE) for all 16 Reference Picture Addresses (RefAddr[0-15])** **Value** / **Name** 11b – Good chance of generating hits. 10b – Next good chance of generating hits 01b – Decent chance of generating hits 00b – Poor chance of generating hits
52	31:6	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address** Project: CHV, BSW) Format: GraphicsAddress[31:6] **For decoder:** Specifies the 64 byte aligned buffer address for writing a single error/status record into the memory when **Pic Error/Status Report Enable** is set in the MFX_PIPE_MODE_SELECT Command. The error/status record is written by HW at the end of decoding one single picture. The record is written in a fixed format, total 96-bits in size always. Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format. **For encoder:** Specifies the 64 byte aligned buffer address for reading the per-MB indirect data from memory when **MacroblockStatEnable** is set in the MFX_AVC_IMG_STATE Command. This field is used for dynamic repeat of frame in PAK for Rate Control. Also used for feeding coding information back to the Host, Video Preprocessing Unit, and ENC Unit. All data are written in fixed formats, and therefore all record sizes are known in the hardware. Hardware can calculate the offset into this base address for per-MB data.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
53	31:16	**Reserved** Format: MBZ
	15:0	**Macroblock Buffer Base Address or Decoded Picture Error/Status Buffer Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Macroblock Status Buffer Base Address
54	31:15	**Reserved** Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved**

MFX_PIPE_BUF_ADDR_STATE

	Format:	MBZ

10:9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 Macroblock Status Buffer - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Macroblock Status Buffer Base Address

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 Target Cache (TC) for Macroblock Status Buffer Base Address

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 Reserved

Project:	CHV, BSW

1:0 Age for QUADLRU (AGE) for Macroblock Status Buffer Base Address

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC.If a particular

MFX_PIPE_BUF_ADDR_STATE

allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

55 — 31:6 — Macroblock ILDB StreamOut Buffer Base Address

Format:	GraphicsAddress[31:6]

Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when **Debocker streamout enable** is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.

55 — 5:0 — Reserved

Project:	CHV, BSW
Format:	MBZ

56 — 31:16 — Reserved

Format:	MBZ

56 — 15:0 — Macroblock ILDB StreamOut Buffer Base Address High

Project:	CHV, BSW
Format:	GraphicsAddress[47:32]

This field is for the upper range of Deblocking Filter Row Store Scratch Address

57 — 31:15 — Reserved

Format:	MBZ

57 — 14:13 — Reserved

Project:	CHV, BSW
Format:	MBZ

57 — 12:11 — Reserved

Format:	MBZ

57 — 10:9 — Reserved

Project:	CHV, BSW
Format:	MBZ

57 — 8:7 — Macroblock ILDB StreamOut Buffer - Arbitration Priority Control

Project:	CHV, BSW

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

MFX_PIPE_BUF_ADDR_STATE

		Value / **Name** 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Macroblock ILDB StreamOut Buffer Base Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** / **Description** 00b — Use Cacheability Controls from page table 01b — UC — Uncacheable - non-cacheable 10b — WT — Writethrough 11b — WB — Writeback
	4:3	**Target Cache (TC) for Macroblock ILDB StreamOut Buffer Base Address** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching. **Value** / **Name** 00b — eLLC Only - not snooped in GT 01b — LLC Only 10b — LLC/eLLC Allowed 11b — L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Macroblock ILDB StreamOut Buffer Base Address** Project: CHV, BSW This field allows the selection of AGE parameter for a given surface in LLC or eLLC.If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. **Value** / **Name** 11b — Good chance of generating hits. 10b — Next good chance of generating hits 01b — Decent chance of generating hits 00b — Poor chance of generating hits
58	31:6	**Second Macroblock ILDB StreamOut Buffer Base Address** Format: GraphicsAddress[31:6]

MFX_PIPE_BUF_ADDR_STATE

		64 byte aligned buffer. Specifies the 64 byte aligned buffer address for writing MB ILDB parameter per MB to memory when Debocker streamout enable is set in the MFX_PIPE_MODE_SELECT Command. The ildb MB control parameters are written by HW at the end of each decoding MB. Only AVC edge information is being streamed out. It is used in AVC decode mode only.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
59	31:16	**Reserved** Format: MBZ
	15:0	**Second Macroblock ILDB StreamOut Buffer Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field is for the upper range of Second Macroblock ILDB StreamOutBuffer Base Address.
60	31:15	**Reserved** Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12:11	**Reserved** Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Second Macroblock ILDB StreamOut Buffer - Arbitration Priority Control** Project: CHV, BSW This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Second Macroblock ILDB StreamOut Buffer Base Address** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable

MFX_PIPE_BUF_ADDR_STATE

10b	WT	Writethrough
11b	WB	Writeback

4:3 Second Macroblock ILDB StreamOut Buffer Base Address - Target Cache (TC)

Project:	CHV, BSW

This field allows the choice of LLC vs eLLC for caching

Value	Name	Description
00b	eLLC Only	not snooped in GT
01b	LLC Only	
10b	LLC/eLLC Allowed	
11b	L3, LLC, eLLC Allowed	

2 Reserved

1:0 Age for QUADLRU (AGE) for Second Macroblock ILDB StreamOut Buffer Base Address

Project:	CHV, BSW

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
00b	Poor chance of generating hits
01b	Decent chance of generating hits
10b	Next good chance of generating hits
11b	Good chance of generating hits

MFX_PIPE_MODE_SELECT

MFX_PIPE_MODE_SELECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

Specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis.
The MFX_PIPE_MODE_SELECT command specifies which codec and hardware module is being used to encode/decode the video data, on a per-frame basis. It also configures the hardware pipeline according to the active encoder/decoder operating mode for encoding/decoding the current picture. Commands issued specifically for AVC and MPEG2 are ignored when VC1 is the active codec.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_COMMON Format: OpCode
	26:24	**Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpA** Default Value: 0h Format: OpCode
	20:16	**SubOpB** Default Value: 0h MFX_PIPE_MODE_SELECT Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 3h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31	**Reserved**
	30	**Reserved** Project: CHV, BSW

MFX_PIPE_MODE_SELECT

29	**Reserved**
28:27	**Reserved**
26	**Reserved**
25	**Reserved**
24	**Reserved**
23:19	**Reserved**
18	**Extended stream out enable** Format: U1 This bit can be set only when VDEnc_Mode is set. When this bit is set and MB stream out is enabled, per MB 1CL of data is streamed out. The actual contents of the stream out are listed in Media VDBOX > Encoder VDEnc mode StreamOut Data Structure Definition. When this bit is not set, per MB ¼ CL data is streamed out. The actual contents of the stream out are listed in Media VDBOX > Encoder StreamOut Mode Data Structure Definition.
17	**Decoder Short Format Mode** For IT mode, this bit must be 0.

Value	Name	Description
1	Long Format Driver Interface	[CHV, BSW] AVC/VC1/MVC/ VP8 Long Format Mode is in use.
0	Short Format Driver Interface **[Default]**	AVC/VC1/MVC/ VP8 Short Format Mode is in use **Note: There is no Short Format for VP8 yet, so this field must be set to 1 for VP8.**

16:15	**Decoder Mode select** Each coding standard supports two entry points: VLD entry point and IT (IDCT) entry point. This field selects which one is in use.This field is only valid if Codec Select is 0 (decoder).

Value	Name	Description
0h	VLD Mode	All codec minimum must support this mode Configure the MFD Engine for VLD Mode Note: All codec minimum must support this mode
1h	IT Mode	Configure the MFD Engine for IT Mode Note: Only VC1 and MPEG2 support this mode
2h	Deblocker Mode	Configure the MFD Engine for Standalone Deblocker Mode. Require streamout AVC edge control information from preceeding decoding pass.

14:13	**Reserved**
12	**Deblocker Stream-Out Enable** Project: CHV, BSW This field indicates if Deblocker information is going to be streamout during VLD decoding. For AVC, it is needed to enable the deblocker streamout as the AVC Disable_DLKFilterIdc is a slice

MFX_PIPE_MODE_SELECT

level parameters. Driver needs to determine ahead of time if at least one slice of the current frame/ has deblocker ON.

Value	Name	Description
0h	Disable	Disable streamout of deblocking control information for standalone deblocker operation.
1h	Enable	

11 **Pic Error/Status Report Enable.**

Project:	CHV, BSW

This field control whether the error/status reporting is enable or not.0: Disable1: EnableIn decoder modes: Error reporting is written out once per frame. The Error Report frame ID listed in DW3 along with the VLD/IT error status bits are packed into one cache and written to the "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command. Note: driver shall program different error buffer addresses between pictrues; otherwise, hardware might overwrite previous written data if driver does not read it fast enough.In encoder modes: Not used
Please refer to "Media VDBOX -> Video Codec -> Other Codec Functions -> MFX Error Handling -> Decoder" session for the output format.

Value	Name
0h	Disable
1h	Enable

10 **Stream-Out Enable**
This field controls whether the macroblock parameter stream-out is enabled during VLD decoding for transcoding purpose.

Value	Name
0h	Disable
1h	Enable

Programming Notes
In decoder modes: The Stream-Out feature is added to support transcoding. While decoding the input compressed stream, selected decoded information may be used by the encoder for re-compression.In encoder modes: This feature used to perform dynamic Multipass of PAK for conformance pupose. Also it provides feedback to host (ENC) for future needs. Software can use this bit to disable writing PAK steam data to the streamout buffer for last pass of frame in PAK. Thus, save memory bandwidth.

9 **Post Deblocking Output Enable (PostDeblockOutEnable)**
This field controls the output write for the reconstructed pixels AFTER the deblocking filter.In MPEG2 decoding mode, if this is enabled, VC1 deblocking filter is used.

Value	Name
0h	Disable
1h	Enable

8 **Pre Deblocking Output Enable (PreDeblockOutEnable)**

MFX_PIPE_MODE_SELECT

This field controls the output write for the reconstructed pixels BEFORE the deblocking filter.

Value	Name
0h	Disable
1h	Enable

7:6 **Reserved**

Project:	CHV, BSW
Format:	MBZ

5 **Stitch Mode**

Exists If:	//CodecSel=Encode and StandardSel=AVC

Value	Name	Description
0h	Not in stitch mode	
1h	In the special stitch mode	This mode can be used for any Codec as long as bitfield conditions are met.

4 **Codec Select**

Value	Name	Description
0h	Decode	
1h	Encode	Valid only if StandardSel is AVC or MPEG2)

3:0 **Standard Select**

Value	Name	Description
0000b	MPEG2	
0001b	VC1	
0010b	AVC	Covers both AVC and MVC
0011b	JPEG	
0100b	Reserved	
0101b	VP8	Decoder and Encoder
0110b	Reserved	
0111b	Reserved	
1111b	UVLD	SW decoder w/ embedded micro-controller and co-processor

2 31 **Reserved**

Format:	MBZ

30 **Reserved**

Project:	CHV, BSW

29 **Reserved**

Format:	MBZ

28 **Reserved**

MFX_PIPE_MODE_SELECT

27	**Reserved**
26	**Reserved** Project: CHV, BSW
25	**Reserved** Project: CHV, BSW
24	**VHR MVC Field Reference List Logic Enable** Project: CHV, BSW 0 – Disable **[Default]** – Disable MVC Field Logic 1 – Enable – VHR MVC Field Enable
23	**Reserved**
22:21	**Reserved**
20:19	**Reserved** Project: CHV, BSW Format: MBZ
18	**Reserved** Format: MBZ
17	**Reserved** Project: CHV, BSW
16	**Reserved**
15	**Reserved**
14	**VLF 720i (Odd Height) in VC1 Mode** Project: CHV, BSW This bit indicates VLF write out VC1 picture with odd height (in MBs). 0 – Disable **[Default]** 1 – Enable – 720i Enable
13	**Reserved**
12	**Reserved**
11	**Reserved**
10	**MPC pref08x8_disable Flag (Default 0)** 0 – Disable 1 – Enable
9	**Reserved**

MFX_PIPE_MODE_SELECT

		Format: MBZ
	8	**Reserved** Project: CHV, BSW
	7	**Reserved**
	6	**Clock gate Enable at Slice-level** BitFieldDesc: Value / Name / Description 0 / Disable / Disable Slice-level Clock gating, Unit-level Clock gating will apply 1 / Enable / Enable Slice-level Clock gating, overrides any Unit level Clock gating
	5	**Reserved**
	4	**Reserved**
	3	**VDS ILDB Calculation** Project: CHV, BSW This bit forces all MB into INTRA MBs before doing ILDB control generation in VDS. Value / Name / Description 0 / Disable **[Default]** / Use original definition for ILDB calculation. 1 / Enable / Force neighbor Intra MB = 1 on ILDB BS calculation. **Programming Notes** When the bit is '0', the ILDB control generation will be the same as the original spec (AVC/VC1).
	2:1	**Reserved** Project: CHV, BSW
	0	**Reserved** Project: CHV, BSW
3	31:0	**Pic Status/Error Report ID** Exists If: //Decoder Mode Only Format: U32 In decoder modes: Error reporting is written out once per frame. This field along with the VLD error status bits are packed into one cache and written to the memory location specified by "Decoded Picture Error/Status Buffer address" listed in the MFX_PIPE_BUF_ADDR_STATE Command. Value / Name / Description 0h / 32-bit unsigned / Unique ID Number 1h / Reserved /
4	31:0	**Media Soft-Reset Counter (per 1000 clocks)** Project: CHV, BSW In decoder modes, this indicates the number of clocks (per 1000) VINunit will wait for inactivity from MFX pipeline before issuing media soft reset. If this counter is set to 0, VINunit will never

MFX_PIPE_MODE_SELECT

		issue soft media reset. In encoder modes: This counter must be set to 0 to disable media soft reset since encoder mode is not supported.

Value	Name	Project
0	Disable	CHV, BSW

MFX_QM_STATE

MFX_QM_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This is a common state command for AVC encoder modes. For encoder, it represents both the forward QM matrices as well as the decoding QM matrices.This is a Frame-level state. Only Scaling Lists specified by an application are being sent to the hardware. The driver is responsible for determining the final set of scaling lists to be used for decoding the current slice, based on the AVC Spec Table 7-2 (Fall-Back Rules A and B).In MFX AVC PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order. But the Forward Q scaling lists are sent in column-wise raster order (column-by-column) to simplify the H/W. Driver will perform all the scan order conversion for both ForwardQ and IQ.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_MULTI_DW Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 7h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 20h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:2	**Reserved** Format: MBZ

MFX_QM_STATE

	1:0	**AVC** Exists If: //AVC- Decoder Only **For AVC QM Type**: This field specifies which Quantizer Matrix is loaded. 0 – AVC_4x4_Intra_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 1 – AVC_4x4_Inter_MATRIX, (Y-4DWs, Cb-4DWs, Cr-4DWs, reserved-4DWs) 2 – AVC_8x8_Intra_MATRIX 3 – AVC_8x8_Inter_MATRIX
	1:0	**MPEG2** Exists If: //MPEG2- Decoder Only **For MPEG2 QM Type**: This field specifies which Quantizer Matrix is loaded. 0 – MPEG_INTRA_QUANTIZER_MATRIX 1 – MPEG_NON_INTRA_QUANTIZER_MATRIX 2-3 – Reserved
	1:0	**JPEG** Project: CHV, BSW Exists If: //JPEG- Encoder Only **For JPEG QM Type**:This field specifies which Quantizer Matrix is loaded. 0 – JPEG_Luma_Y_QUANTIZER_MATRIX (or R) 1 – JPEG_Chroma_Cb_QUANTIZER_MATRIX (or G) 2 – JPEG_Chroma_Cr_QUANTIZER_MATRIX (or B) **Programming Notes** For JPEG encoder, each quantization element presents 16-bit 1/QM[i][j]. In RGB encoding, because the order input image components can be RGB, GBR, BGR, YUV, the value 0 is used for the first image component, the value 1 is used for the second image component, and the value 2 is used for the third image component.
2..33	31:0	**Forward Quantizer Matrix** Project: All Format: U32 The format of a Quantizer Matrix is an 8x8 matrix in raster order. Each element is an unsigned byte.

MFX_STATE_POINTER

MFX_STATE_POINTER

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

The MFX_STATE_POINTER command, issued at picture level, is used to set up the indirect pointers for VCS to fetch all the MFX states (Image state, Slice state, etc.) needed for the encoding/decoding process in PAK/IT mode. The encoding/decoding states are presented by state commands, which are grouped into separate sets (picture level, slice level, etc.), and each is stored in its own memory buffer referred by an indirect state pointer. The content of each indirect state buffer is a list of MFX state commands with no special format requirements. The sequence of commands in each indirect state buffer is terminated by a MI_BATCH_BUFFER_END command (acts as the last command marker). Therefore, indirect state buffers can have different and variable length of command sequences.

The indirection is designed to facilitate context switching in the middle of a codec operation. The smallest granularity of interruption is designed to be at a completed MB row in AVC/VC1/MPEG2 IT and AVC PAK operating modes as well as in VC1/MPEG2 VLD mode. There is no support for context switch in AVC VLD mode.

Hardware supports up to 4 separate indirect state pointers, allowing software to manage the grouping of state commands. During context switch, hardware restores (re-issues) the latest version of each indirect state pointer, if present.

MFX_STATE_POINTER command can only program one indirect state pointer at a time. MI_FLUSH will invalidate all indirect state buffer pointers inside VCS.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFX_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON_STATE Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 6h Format: OpCode
	15:12	**Reserved**

MFX_STATE_POINTER

	11:0	**DWord Length** Default Value: 0h DWORD_COUNT_n Project: All Format: =n Total Length - 2
1	31:5	**State Pointer** Format: GeneralStateOffset[31:5]Indirect State Buffer Specifies the 32-byte aligned address of an Indirect State Buffer. This pointer is relative to the General State Base Address.
	4:2	**Reserved** Project: All Format: MBZ
	1:0	**State Pointer Index** Specifies one of the four indirect state pointers to program.

Value	Name	Description	Project
00b		indirect state pointer 0 (image state)	All
01b		indirect state pointer 1 (slice state)sc	All
10b		indirect state pointer 2	
11b		indirect state pointer 3	

MFX_STITCH_OBJECT

MFX_STITCH_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MFC_STITCH_OBJECT command is used when stitch-enabled is set to 1, while CodecSel and StandardSel are set to ENCODE and AVC, respectively. This command is used, for example, to stitch multiple bitstreams to form a transport stream.
It is a variable length command as the data to be inserted are presented as either inline data and/or indirect data of this command. Multiple insertion commands can be issued back to back in a series. It is host software's responsibility to make sure their corresponding data will properly stitch together to form a valid output. Hardware keeps track of an output bitstream buffer current byte position and the associated next bit insertion position index. Context switch interrupt is not supported by this command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFC_STITCH_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 0h MFX_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: Ah Format: OpCode
	15:12	**Reserved**
	11:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = Variable Length in DW (>= 3) Format: =n Total Length - 2 If it is 3, it indicates the absent of inline data.
1	31:18	**Reserved**
	17:16	**Source Data Starting Byte Offset** Source Data Starting Byte Position within the very first inline DW.

MFX_STITCH_OBJECT

	15:14	**Reserved** Format: MBZ
	13:8	**Source Data Ending Bit Inclusion** Source Data to be included in the very last inline DW. Follows the MSBit is the upper bit of each byte within the DW. The lower byte is actually processed first.For example, SrCDataEndingBitInclusion =9, bit 7:0 and bit 15 are included as valid header data. Value: [1,32] — Name:
	7:4	**Reserved**
	3	**Reserved**
	2	**Last Source Header Data Insert Command Flag** To process a series of consecutive insertion commands, this flag (=1) indicates the current command is the last 'header' insertion in the series.In CABAC, hardware must perform the "1" insert for byte align for Slice Header before Slice Data comes in in the next PAK-OBJECT command.In CAVLC, hardware ignores this bit.
	1	**Last Destination Data Insert Command Flag** THIS FIELD MUST BE THE SAME AS Last Source Header Data Insert Command Flag No more insertion command and no more PAK-OBJECT command follows.Flush data out to memory
	0	**Reserved**
2	31:19	**Reserved** Format: MBZ
	18:0	**Indirect Data Length** Project: CHV, BSW Format: U19 This field provides the length in bytes of the indirect data. A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect Data Start Address field is ignored. This field must have the same alignment as the Indirect Object Data Start Address.
3	31:0	**Indirect Data Start Address** Format: MfxIndirectBitstreamObjectAddress[31:0] This field specifies the Graphics Memory starting address of the data to be loaded into the kernel for processing. This pointer is relative to the MFX Indirect Bitstream Object Base Address. Hardware ignores this field if indirect data is not present.
4..n	31:0	**Insert Data PayLoad** Inline data to be inserted to the output bitstream buffer

MFX_SURFACE_STATE

MFX_SURFACE_STATE

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

This command is common for all encoding/decoding modes, to specify the uncompressed YUV picture (i.e. destination surface) or intermediate streamout in/out surface (e.g. coefficient/residual) (field, frame or interleaved frame) format for reading and writing:

- Uncompressed, original input picture to be encoded
- Reconstructed non-filtered/filtered display picturec(becoming reference pictures as well for subsequent temporal inter-prediction)

Since there is only one media surface state being active during the entire encoding/decoding process, all the uncompressed/reconstructed pictures are defined to have the same surface state. The primary difference among picture surface states is their individual programmed base addresses, which are provided by other state commands and not included in this command. MFX engine is making the association of surface states and corresponding buffer base addresses.
MFX engine currently supports only one media surface type for video and that is the NV12 (Planar YUV420 with interleaved U (Cb) and V (Cr). For optimizing memory efficiency based on access patterns, only TileY is supported. For JPEG decoder, only IMC1 and IMC3 are supported. Pitch can be wider than the Picture Width in pixels and garbage will be there at the end of each line. The following describes all the different formats that are supported and not supported in Gen7 MFX :

- NV12 - 4:2:0 only; UV interleaved; Full Pitch, U and V offset is set to 0 (the only format supported for video codec); vertical UV offset is MB aligned; UV xoffsets = 0. JPEG does not support NV12 format because non-interleave JPEG has performance issue with partial write (in interleaved UV format)
- IMC 1 & 3 - Full Pitch, U and V are separate plane; (JPEG only; U plane + garbage first in full pitch followed by V plane + garbage in full pitch). U and V vertical offsets are block aligned; U and V xoffset = 0; there is no gap between Y, U and V planes. IMC1 and IMC3 are different by a swap of U and V. This is the only format supported in JPEG for all video subsampling types (4:4:4, 4:2:2 and 4:2:0)
- We are not supporting IMC 2 & 4 - Full Pitch, U and V are separate plane (JPEG only; U plane first in full pitch followed by V plane in full pitch - U and V plane are side-by-side). U and V vertical offsets are 16-pixel aligned; V xoffset is half-pitch aligned; U xoffset is 0; there is no gap between Y, U and V planes. IMC2 and IMC4 are different by a swap of U and V.
- We are not supporting YV12 - half pitch for each U and V plane, and separate planes for Y, U and V (U plane first in half pitch followed by V plane in half pitch). For YV12, U and V vertical offsets are block aligned; U and V xoffset = 0; there is no gap between Y, U and V planes

Note that the following data structures are not specified through the media surface state

- 1D buffers for row-store and other miscellaneous information.
- 2D buffers for per-MB data-structures (e.g. DMV biffer, MB info record, ILDB Control and Tcoeff/Stocoeff).

This surface state here is identical to the Surface State for deinterlace and sample_8x8 messages described in the Shared Function Volume and Sampler Chapter.
For non pixel data, such as row stores, indirect data (Compressed Slice Data, AVC MV record, Coeff record and

MFX_SURFACE_STATE

AVC ILDB record) and streamin/out and output compressed bitstream, a linear buffer is employed. For row stores, the H/W is designed to guarantee legal memory accesses (read and write). For the remaining cases, indirect object base address, indirect object address upper bound, object data start address (offset) and object data length are used to fully specified their corresponding buffer. This mechanism is chosen over the pixel surface type because of their variable record sizes.
All row store surfaces are linear surface. Their addresses are programmed in Pipe_Buf_Base_State or Bsp_Buf_Base_Addr_State

Programming Notes

VC1 I picture scaling: Even though VC1 allows I reconstructed picture scaling (via RESPIC), as such scaling is only allowed at I picture. All subsequent P (and B) pictures must have the same picture dimensions with the preceding I picture. Therefore, all reference pictures for P or B picture can share the same surface state with the current P and B picture. Note : H/W is not processing RESPIC. Application is no longer expecting intel decoder pipelineand kernel to perform this function, it is going to be done in the video post-processing scaler or display controller scale as a separate step and controller.

All video codec surfaces must be NV12 Compliant, except JPEG. U/V vertical must be MB aligned for all video codec (further contrained for field picture), but JPEG can be block aligned. All video codec and JPEG uses Tiled - Y format only, for uncompressed pixel surfaces.

Even for JPEG planar 420 surface, application may provide only 1 buffers, but there is still only one single surface state for all of them. If IMC equal to 1, 2, 3 or 4, U and V have the pitch same as Y. And U and V will have different offset, each offset is block aligned.

DWord	Bit	Description	
0	31:29	**Command Type**	
		Default Value:	3h PARALLEL_VIDEO_PIPE
		Format:	OpCode
	28:27	**Pipeline**	
		Default Value:	2h MFX_COMMON
		Format:	OpCode
	26:24	**Opcode**	
		Default Value:	0h MFX_COMMON_STATE
		Format:	OpCode
	23:21	**SubOpA**	
		Default Value:	0h
		Format:	OpCode
	20:16	**SubOpB**	
		Default Value:	1h
		Format:	OpCode
	15:12	**Reserved**	
		Format:	MBZ

MFX_SURFACE_STATE

	11:0	**DWord Length** Format: =n Total Length - 2 **Value** / **Name** / **Description** 4h / DWORD_COUNT_n **[Default]** / Excludes DWord (0,1)
1	31:4	**Reserved** Format: MBZ
	3:0	**Surface Id** Project: CHV, BSW Format: U4 **Value** / **Name** / **Description** 0000b / Reserved / 0001b / Reserved / 0010b / Reserved / 0011b / Reserved / 0100b / Source Input Picture (encoder) / 8-bit uncompressed data 0101b / Reconstructed Scaled Reference Picture / 8-bit data
2	31:18	**Height** Format: U14-1 Height This field specifies the height of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. Note : Gen7 Video Codecs must program less than and equal to 4K.(In future, it will be ideal to have this field define in a WORD boundary.)AVC - multiple of 2 MB rows for field pictureVC1 - mulitple of 4 pixels for field pictureMPEG2 - multiple of 2 MB rows for field picJPEG - mulitple of integral MCU (8 or 16 pixels) per picture **Value** / **Name** / **Description** [0,16383] / / representing heights [1,16384] **Programming Notes** • For AVC : For frame picture is a multiple of 16; for field picture is a multiple of 32 • For VC1 : For progressive frames, the frame height and frame width is a multiple of 2 pixels. For interlaced frames, the frame height shall be a multiple of 4 pixels, and its width is a multiple of 2 pixels, based on a PLANAR_420 surface.
	17:4	**Width** Format: U14-1 Width This field specifies the width of the Picture in units of pixels/residuals. For PLANAR surface formats, this field indicates the width of the Y (luma) plane. **Value** / **Name** / **Description**

MFX_SURFACE_STATE

		[0,16383]		representing widths [1,16384]

Programming Notes

- The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field).
- Width (field value + 1) must be a multiple of 2 for PLANAR_420
- MFX HW does not use this field, the picture width is read from IMG State instead, because this field may not equal to the actual picture width. This field is used by the KMD to allocate surface in GTT.

	3:2	**Reserved** Format: MBZ
	1:0	**Cr(V)/Cb(U) Pixel Offset V Direction** Project: All Format: U0.2 exactly as shown in the original spec Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction **Programming Notes** This field is ignored for all formats except PLANAR_420_8
3	31:28	**Surface Format** Format: U4 Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.Usage: For 420 planar YUV surface, use 4; for monochrome surfaces, use 12. For monochrome surfaces, hardware ignores control fields for Chroma planes.This field must be set to 4 - PLANAR_420_8, or 12 - Y8_UNORMNot used for MFX, and is ignored. But for JPEG decoding, this field should be programmed to the same format as JPEG_PIC_STATE. For video codec, it should set to 4 always.

Value	Name	Description
0	YCRCB_NORMAL	
1	YCRCB_SWAPUVY	
2	YCRCB_SWAPUV	
3	YCRCB_SWAPY	
4	PLANAR_420_8	(NV12, IMC1,2,3,4, YV12)
5	PLANAR_411_8	Deinterlace Only
6	PLANAR_422_8	Deinterlace Only
7	STMM_DN_STATISTICS	Deinterlace Only
8	R10G10B10A2_UNORM	Sample_8x8 Only
9	R8G8B8A8_UNORM	Sample_8x8 Only
10	R8B8_UNORM (CrCb	Sample_8x8 Only

MFX_SURFACE_STATE

11	R8_UNORM (Cr/Cb)	Sample_8x8 Only
12	Y8_UNORM	Sample_8x8 Only

27 Interleave Chroma

Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.For AVC/VC1/MPEG VLD and IT modes : set to Enable to support interleave U/V only.For JPEG : set to Disable for all formats (including 4:2:0) - because JPEG does not support NV12. (This field is needed only if JPEG will support NV12; otherwise is ignored.)

Value	Name
1	Enable
0	Disable

26:20 Reserved

Format:	MBZ

19:3 Surface Pitch

Format:	U17-1 pitch in Bytes

This field specifies the surface pitch in (#Bytes).

Value	Name	Description
[0,2047]		to [1B, 2048B]

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile width (i.e.128 bytes aligned). If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.For Y-tiled surfaces: Range = [127, 524287] to [128B,256KB] = [1 tile, 2048 tiles]

2 Half Pitch for Chroma

Format:	Enable

(This field must be set to Disable)This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field. This field is only used for PLANAR surface formats.This field is igored by MFX (unless we support YV12)

1 Tiled Surface

Format:	Boolean

(This field must be set to TRUE: Tiled)This field specifies whether the surface is tiled.This field is ignored by MFX

Value	Name	Description
0	False	Linear
1	True	Tiled

MFX_SURFACE_STATE

		Programming Notes Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory.The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
	0	**Tile Walk** Format: 3D_Tilewalk (This field must be set to 1: TILEWALK_YMAJOR)This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See Memory Interface Functions for details on memory tiling and restrictions.This field is ignored when the surface is linear.This field is ignored by MFX. Internally H/W is always treated this set to 1 for all video codec and for JPEG. Value / Name / Description: 0h / XMAJOR / TILEWALK_XMAJOR 1h / YMAJOR / TILEWALK_YMAJOR **Programming Notes** The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit
4	31	**Reserved** Format: MBZ
	30:16	**X Offset for U(Cb)** Project: All Format: U15 Pixel Offset This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. This field must be set to zero.X Offset for U(Cb) in pixel (This field must be zero for NV12 and IMC 1 and 3) **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be zero.
	15	**Reserved**
	14:0	**Y Offset for U(Cb)** Project: All Format: U15 Pixel Row Offset This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. This field is only used for PLANAR surface formats. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must be multiple of 16 pixels - i.e.

MFX_SURFACE_STATE

		multiple MBs. For JPEG, this field must be a multiple of 16 pixels.
5	31:29	**Reserved**
	28:16	**X Offset for V(Cr)** Format: U13 Offset in Pixels This field must be zero for NV12 and IMC 1 and 3 This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.
	15:0	**Y Offset for V(Cr)** Format: U16 Row Offset in Pixels This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with Interleave Chroma disabled. This field is ignored by all video codec, only used by JPEG. **Programming Notes** For PLANAR_420 surface formats, this field must be multiple of 16 pixels - i.e. multiple MBs. For JPEG, this field must be a multiple of 16 pixels.

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_DIRECTMODE_STATE

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2
Exists If:	//VC1 decoding in VLD modes

This is a picture level command and should be issued only once, even for a multi-slices picture. There is only one DMV buffer for read (when processing a B-picture) and one for write (when processing a P-Picture). Each DMV record is 64 bits per MB, to store the top and bottom field MVs (32-bit MVx,y each).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VC1_DIRECTMODE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved**
	11:0	**DWord Length** Default Value: 0005h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:6	**Direct MV Write Buffer Base Address for the Current Picture** This field provides the base address of the DMV write buffer to store the motion vectors decoded in the current picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software.This buffer must be 64-byte cacheline aligned.The write buffer size is 557,056 bytes for 1 frame. Scalable with frame height, but do not scale with frame width as the hardware assumes frame width (in MBs) fixed at 128 (smallest power of 2 value larger than 120 - 1920x1088 screen resolution).This field is only valid for a P picture

MFX_VC1_DIRECTMODE_STATE

	5:0	**Reserved** Project: CHV, BSW Format: MBZ
2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Direct MV Write Buffer Base Address for the Current Picture [47:32]** Project: CHV, BSW This field is for the upper range of Direct MV Write Buffer Base Address for the Current Picture. This field is used for 48-bit addressing.
3 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved**
	12:11	**Reserved**
	10:9	**Reserved**
	8:7	**Direct MV Write Buffer Base Address for the Current Picture - Arbitration Priority Control** Project: CHV, BSW Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Direct MV Write Buffer for the Current Picture** Project: CHV, BSW This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** 00b: Use Cacheability Controls from page table 01b: Uncacheable (UC) - non-cacheable 10b: Writethrough (WT) 11b: Writeback (WB)

MFX_VC1_DIRECTMODE_STATE

	4:3	**Target Cache (TC) for Direct MV Write Buffer for the Current Picture** Project: CHV, BSW This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Direct MV Write Buffer for the Current Picture** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. 11b: Good chance of generating hits. 10b: Next good chance of generating hits 01b: Decent chance of generating hits 00b: Poor chance of generating hits
4	31:6	**Direct MV Read Buffer Base Address for the Reference Picture** This field provides the base address of the DMV buffer for reference picture. It is a private buffer used by the MPR hardware only. Its content is not accessed by software.All these buffers must be 64-byte cacheline aligned.This field is only valid for a B picture.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ
5 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for 64-bit address extension.
	15:0	**Direct MV Read Buffer Base Address for the Current Picture [47:32]** Project: CHV, BSW This field is for the upper range of Direct MV Read Buffer Base Address for the Current Picture. This field is used for 48-bit addressing.
6 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Format: MBZ
	14:13	**Reserved**

MFX_VC1_DIRECTMODE_STATE

	Project:	CHV, BSW
	Format:	MBZ

12:11 Reserved

Project:	CHV, BSW
Format:	MBZ

10:9 Reserved

Project:	CHV, BSW
Format:	MBZ

8:7 Direct MV Read Buffer Base Address for the Current Picture - Arbitration Priority Control

Project:	CHV, BSW
Format:	U2

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Direct MV Read Buffer for the Current Picture

Project:	CHV, BSW

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name
00b	Use Cacheability Controls from page table
01b	Uncacheable (UC) - non-cacheable
10b	Writethrough (WT)
11b	Writeback (WB)

4:3 Target Cache (TC) for Direct MV Read Buffer for the Current Picture

Project:	CHV, BSW

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC 00b: eLLC Only ("00" setting points TC selection to PTE which defaults to eLLC) 01b: LLC Only (Works at the allocation time, later victimization from LLC downgrades the line to eLLC if present). 10b: LLC/eLLC Allowed. 11b: L3, LLC, eLLC Allowed.

MFX_VC1_DIRECTMODE_STATE

		Value / **Name** 00b — eLLC Only - not snooped in GT 01b — LLC Only 10b — LLC/eLLC Allowed 11b — L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Age for QUADLRU (AGE) for Direct MV Read Buffer for the Current Picture** Project: CHV, BSW Format: Enable This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age (“0,1,2”) it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. **Value** / **Name** 11b — Good chance of generating hits. 10b — Next good chance of generating hits 01b — Decent chance of generating hits 00b — Poor chance of generating hits.

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_PRED_PIPE_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This command is used to set the operating states of the MFD Engine beyond the BSD unit. It is used with both VC1 Long and Short format.Driver is responsible to take the intensity compensation enable signal, the LumScale and the LumShift provided from the DXVA2 VC1 interface, and maintain a history of these values for reference pictures. Together with these three parameters specified for the current picture being decoded, driver will derive and supply the above sets of LumScaleX, LumShiftX and intensity compensation enable (single or double, forward or backward) signals. H/W is responsible to take these state values, and use them to build the lookup table (including the derivation of iScale and iShift) for remapping the reference frame pixels, as well as perfoming the actual pixel remapping calculations/process.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VC1_PRED_PIPE_STATE Format: OpCode
	26:24	**Media Command Opcode** Default Value: 2h VC1_COMMON Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 1h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 0004h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:16	**Reserved**

MFX_VC1_PRED_PIPE_STATE

	15:14	**vin_intensitycomp_Double_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	13:12	**vin_intensitycomp_Double_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	11:10	**vin_intensitycomp_Single_FWDen** Format: U2 for forward reference picture only, to enable top or/and bottom of the reference field enable for single compensation. For frame, may only need one bit. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	9:8	**vin_intensitycomp_Single_BWDen** Format: U2 for backward reference picture only, no double for backward reference. This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:4	**Reference Frame Boundary Replication Mode** Format: U4 This is a bit field with each bit indicating the corresponding picture's boundary replication mode. Bit 11: reference 3 Bit 10: reference 2 Bit 9: reference 1 Bit 8: reference 0 0 = progressive frame replication 1 = interlace frame replication This field is maintained and provided by driver for both long and short VC1 interface format.
	3:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ

MFX_VC1_PRED_PIPE_STATE

	29:24	**LumShift2- single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - Single - FWD** Format: U6 This field is maintained and provided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
3	31:30	**Reserved** Format: MBZ
	29:24	**LumShift2- double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.

MFX_VC1_PRED_PIPE_STATE

	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - double -FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved**
	13:8	**LumScale2 - double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved**
	5:0	**LumScale1 - double - FWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
4	31:30	**Reserved**
	29:24	**LumShift2- single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved**

MFX_VC1_PRED_PIPE_STATE

	13:8	**LumScale2 - single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - Single - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
5	31:30	**Reserved** Format: MBZ
	29:24	**LumShift2- double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	23:22	**Reserved** Format: MBZ
	21:16	**LumShift1 - double -BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.
	15:14	**Reserved** Format: MBZ
	13:8	**LumScale2 - double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.

MFX_VC1_PRED_PIPE_STATE

	7:6	**Reserved** Format: MBZ
	5:0	**LumScale1 - double - BWD** Format: U6 This field is maintained andprovided by driver for both long and short VC1 interface format. And is derived from the intensity compensation enable flag, wBitstreamPCEelement and wBitstreamFcodes parameters provided by the DXVA2 VC1 interface to the driver for each current picture.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 3h MFX_VP8_BSP_BUF_BASE_ADDR_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Value / Name / Description 000h / Excludes DWord (0,1) **[Default]** / A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 008h / / Used for normal encode mode
1	31:6	**Frame Header Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format. 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Frame Header Base Addr - Upper Range**
3	31:9	**Reserved** Format: MBZ
	8:7	**Frame Header Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 **for FrameHeaderBaseAddr - LLC/eLLC Cacheability Control (LeLLCCC)**
This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Cacheable	Use Cacheability Controls from page table
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 **Frame Header Base Addr - Target Cache (TC)**
This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

1:0 **Frame Header Base Addr - Age for QUADLRU (AGE)**
This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
00b	Good chance of generating hits

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		01b: Next good chance of generating hits 10b: Decent chance of generating hits 11b: Poor chance of generating hits
4	31:6	**Intermediate Buffer Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:0	**Intermediate Buffer Base Addr - Upper Range**
6	31:9	**Reserved** Format: MBZ
	8:7	**Intermediate Buffer Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value: 00b – Name: Highest priority Value: 01b – Name: Second highest priority Value: 10b – Name: Third highest priority Value: 11b – Name: Lowest priority
	6:5	**Intermediate Buffer Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. Value: 00b – Name: Cacheable – Description: Use Cacheability Controls from page table Value: 01b – Name: UC – Description: Uncacheable - non-cacheable Value: 10b – Name: WT – Description: Writethrough Value: 11b – Name: WB – Description: Writeback
	4:3	**Intermediate Buffer Base Addr- Target Cache (TC)** This field allows the choice of LLC vs. eLLC for caching Value: 00b – Name: eLLC Only - not snooped in GT Value: 01b – Name: LLC Only Value: 10b – Name: LLC/eLLC Allowed

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Intermediate Buffer Base Addr- Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. Value / Name: 00b: Good chance of generating hits 01b: Next good chance of generating hits 10b: Decent chance of generating hits 11b: Poor chance of generating hits
7	31:0	**Intermediate Buffer Partition-1 Offset** Format: U32 **Programming Notes** All **Intermediate Buffer Partition-[i] Offset** (i = 1 to 8) and **Intermediate Buffer Max Size** need to be cacheline aligned (64Byte aligned).
8	31:0	**Intermediate Buffer Partition-2 Offset** Format: U32
9	31:0	**Intermediate Buffer Partition-3 Offset** Format: U32
10	31:0	**Intermediate Buffer Partition-4 Offset** Format: U32
11	31:0	**Intermediate Buffer Partition-5 Offset** Format: U32
12	31:0	**Intermediate Buffer Partition-6 Offset** Format: U32
13	31:0	**Intermediate Buffer Partition-7 Offset** Format: U32
14	31:0	**Intermediate Buffer Partition-8 Offset** Format: U32
15	31:0	**Intermediate Buffer Max Size** Format: U32
16	31:6	**Final Frame Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:0	**Reserved** Format: MBZ
17	31:16	**Reserved** Format: MBZ
	15:0	**Final Frame Base Addr - Upper Range**
18	31:9	**Reserved** Format: MBZ
	8:7	**Final Frame Base Addr - Arbitration Priority Control** Value: 00b – Name: Highest priority Value: 01b – Name: Second highest priority Value: 10b – Name: Third highest priority Value: 11b – Name: Lowest priority
	6:5	**Final Frame Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. Value: 00b – Name: Cacheable – Description: Use Cacheability Controls from page table Value: 01b – Name: UC – Description: Uncacheable - non-cacheable Value: 10b – Name: WT – Description: Writethrough Value: 11b – Name: WB – Description: Writeback
	4:3	**Final Frame Base Addr - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching Value: 00b – Name: eLLC Only - not snooped in GT Value: 01b – Name: LLC Only Value: 10b – Name: LLC/eLLC Allowed Value: 11b – Name: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Final Frame Base Addr - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		Value / **Name** 00b — Good chance of generating hits 01b — Next good chance of generating hits 10b — Decent chance of generating hits 11b — Poor chance of generating hits
19	31:6	**Reserved** Format: MBZ
	5:0	**FinalFrameByteOffset** Format: U6 Specify byte offset within a 64-byte cacheline where the bitstream should be inserted at.
20	31:6	**Streamout Base Addr** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Streamout Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority
	3	**Reserved**
	2	**Streamout Base Addr - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Streamout Base Addr - Cacheability Control** Format: U2 EnumeratedType This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC) **Value** / **Name** / **Description** 00b — GTT Entry — Use cacheability control bits from GTT entry 01b — Not LLC or MLC — Data is not cached in LLC or MLC 10b — LLC but not MLC — Data is cached in LLC but not MLC

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		11b	Both LLC and MLC	Data is cached in both LLC and MLC
21	31:16	**Reserved**		
		Format:		MBZ
	15:0	**Streamout Base Addr - Upper Range**		
22	31:9	**Reserved**		
	8:7	**Streamout Base Addr - Arbitration Priority Control**		

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

6:5 **Streamout Base Addr - LLC/eLLC Cacheability Control (LeLLCCC)**

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description
00b	Cacheable	Use Cacheability Controls from page table
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 **Streamout Base Addr - Target Cache (TC)**

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

1:0 **Streamout Base Addr - Age for QUADLRU (AGE)**

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
00b	Good chance of generating hits
01b	Next good chance of generating hits
10b	Decent chance of generating hits

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		11b: Poor chance of generating hits
23	31:6	**Coeff Probs StreamIn Surface** Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format 48-bit AbsAddr StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Coeff Probs StreamIn Surface - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. 00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	3	**Reserved**
	2	**Coeff Probs StreamIn Surface - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Coeff Probs StreamIn Surface - Cacheability Control** Format: U2 EnumeratedType This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC) 00b: GTT Entry — Use cacheability control bits from GTT entry 01b: Not LLC or MLC — Data is not cached in LLC or MLC 10b: LLC but not MLC — Data is cached in LLC but not MLC 11b: Both LLC and MLC — Data is cached in both LLC and MLC
24	31:16	**Reserved** Format: MBZ
	15:0	**Coeff Probs StreamIn Surface - Upper Range** This field is for the upper range of Coeff Probs[4][8][3][11]
25	31:9	**Reserved** Format: MBZ
	8:7	**Coeff Probs StreamIn Surface - Arbitration Priority Control** 00b: Highest priority 01b: Second highest priority

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		10b: Third highest priority 11b: Lowest priority
	6:5	**Coeff Probs StreamIn Surface - LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. 00b: Cacheable - Use Cacheability Controls from page table 01b: UC - Uncacheable - non-cacheable 10b: WT - Writethrough 11b: WB - Writeback
	4:3	**Coeff Probs StreamIn Surface - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching 00b: eLLC Only - not snooped in GT 01b: LLC Only 10b: LLC/eLLC Allowed 11b: L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Coeff Probs StreamIn Surface - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 00b: Good chance of generating hits 01b: Next good chance of generating hits 10b: Decent chance of generating hits 11b: Poor chance of generating hits
26	31:6	**Token Statistics Surface** Format: TokenStatisticsAddress[31:6] 48-bit Abs. Address StreamIn Surface **Note:**The format is linear vs. tile for better performance.
	5:4	**Frame Header Base Addr - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. Value | Name

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		00b: Highest priority 01b: Second highest priority 10b: Third highest priority 11b: Lowest priority
	3	**Reserved**
	2	**Token Statistics Surface - Graphics Data Type (GFDT)** Format: U1 This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.
	1:0	**Token Statistics Surface - Cacheability Control** This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC). **Value / Name / Description** 00b / GTT Entry / Use cacheability control bits from GTT entry 01b / Not LLC or MLC / Data is not cached in LLC or MLC 10b / LLC but not MLC / Data is cached in LLC but not MLC 11b / Both LLC and MLC / Data is cached in both LLC and MLC
27	31:16	**Reserved**
	15:0	**Token Statistics Surface** This field is for the upper range of Token Statistics Address.
28	31:9	**Reserved** Format: MBZ
	8:7	**Token Statistics Surface - Arbitration Priority Control** This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value / Name** 00b / Highest priority 01b / Second highest priority 10b / Third highest priority 11b / Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for CoeffProbs StreamIn Surface** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value / Name / Description** 00b / Use Cacheability Controls from page table / 01b / UC / Uncacheable - non-cacheable 10b / WT / Writethrough

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

		11b WB Writeback
	4:3	**Token Statistics Surface - Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching. Value: 00b — eLLC Only - not snooped in GT Value: 01b — LLC Only Value: 10b — LLC/eLLC Allowed Value: 11b — L3, LLC, eLLC Allowed
	2	**Reserved**
	1:0	**Token Statistics Surface - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. Value: 11b — Good chance of generating hits. Value: 10b — Next good chance of generating hits Value: 01b — Decent chance of generating hits Value: 00b — Poor chance of generating hits
29	31:6	**MPC RowStore Surface Address Low** Format: GraphicsAddress[31:6] 48-bit Abs. Address StreamIn/StreamOut Surface. **Note:** The format is linear vs. tile for better performance.
	5:4	**MPC RowStore Base Addr - Arbitration Priority Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value: 00b — Highest priority Value: 01b — Second highest priority Value: 10b — Third highest priority Value: 11b — Lowest priority
	3	**Reserved**
	2	**MPC RowStore Surface Graphics Data Type (GFDT)** Format: U1 **This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by the GTT. The effective GFDT is the logical OR of this field with the GFDT from the GTT entry. This field is ignored for reads.**

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	1:0	**MPC RowStore Surface - Cacheability Control** **This field controls cacheability in the mid-level cache (MLC) and last-level cache (LLC).** Value / Name / Description: 00b – GTT Entry – Use cacheability control bits from GTT entry 01b – Not LLC or MLC – Data is not cached in LLC or MLC 10b – LLC but not MLC – Data is cached in LLC but not MLC 11b – Both LLC and MLC – Data is cached in both LLC and MLC
30	31:16	**Reserved** Format: MBZ
	15:0	**MPC RowStore Surface Address High** Format: GraphicsAddress[47:32] This field is for the upper range of Token Statistics Address.
31	31:9	**Reserved** Format: MBZ
	8:7	**MPC RowStore - Arbitration Priority Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value / Name: 00b – Highest Priority 01b – Second highest Priority 10b – Third highest Priority 11b – Lowest Priority
	6:5	**MPC RowStore - Memory Type: LLC/eLLC Cacheability Control** **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value / Name / Description: 00b – Use Cacheability Controls from page table – 01b – UC – Uncacheable - non-cacheable 10b – WT – Writethrough 11b – WB – Writeback
	4:3	**MPC RowStore - Target Cache** Format: U2 **This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.** Value / Name: 00b – eLLC Only 01b – LLC Only 10b – LLC/eLLC Allowed 11b – L3, LLC, eLLC Allowed

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

	2	**Reserved**
	1:0	**MPC RowStore Surface - Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits.
01b	Decent chance of generating hits.
00b	Poor chance of generating hits.

MFX_VP8_Encoder_CFG

MFX_VP8_Encoder_CFG

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_PIC_STATE.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 2h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 1h MFX_VP8_ENCODER_CFG Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n

Value	Name	Description	Project
000h	Excludes DWord (0,1) **[Default]**	A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware."	
01Ch		Used for normal encode mode	CHV, BSW

MFX_VP8_Encoder_CFG

1	31:11	**Reserved** Format: MBZ
	10	**VBSPunitPowerClock Gating Disable** Project: CHV, BSW Format: U1 VBSPunit Power Clock Gating Disable.
	9	**Compressed Bitstream Output Disable** Project: CHV, BSW Format: U1 Disable Compressed Bitstream Output.**(Both Final Bitstream and Intermediate bit buffer)**
	7	**Per Segment Delta Qindex / LoopFilter Disable** Project: +, CHV, BSW Format: U1 Disable Per Segment Delta Qindex / Loop Filter in Rate Control.
	6	**Rate Control Initial Pass** Project: CHV, BSW Format: U1 Value / Name: 1 – Initial pass 0 – Subsequence Pass(es)
	5	**Skip Final Bitstream when Over / Under flow** Format: U1 Skip Final Bitstream conditionally on Over/Under flow in rate control and intermediate Bit Buffer Overrun.
	4	**Update Segment Feature Data Flag** Exists If: //VP8 Encoder Format: U1 Enable for Frame Header per Segment Quantizer / LoopFilter Update
	3	**Bitstream Statistics Output Enable** Enable Bitstream Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]
	2	**Token Statistics Output Enable** Enable Token Statistics Output at Memory Surface in MFX_VBSP_BUF_ADDR_STATE DW[26:28]
	1	**Final Bitstream Output Disable** Format: U1 Disable Final Bitstream Output.

MFX_VP8_Encoder_CFG

	0	**Performance Counter Enable** Format: U1 Enable Performance Counter in Streamout.
2	31:8	**Reserved** Format: MBZ
	7	**Qindex_Clamp_High_mask for overflow** Project: CHV, BSW Format: U1 If current frame is overflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit1. In another word, subsequent passes would be skipped.
	6	**Qindex_Clamp_High_mask for underflow** Project: CHV, BSW Format: U1 If current frame is underflow and this mask is set, it would mask out MFX_VP8_Img_Status register. DW1.bit0. In another word, subsequent passes would be skipped
	5	**Final Bistream Buffer Overrun Enable Mask** Project: CHV, BSW Format: U1 Enable Final Bitstream Buffer Overrun detection feature.
	4	**Intermediate Bit Buffer Overrun Enable Mask** Format: U1 Enable Intermediate Bit Buffer Overrun detection feature.
	3	**Max Intra MB Bit Count Check Enable Mask** Format: U1 Enable Max. Intra MB bit count check in Streamout.
	2	**Max Inter MB Bit Count Check Enable Mask** Format: U1 Enable Max. Inter MB bit count check in Streamout.
	1	**Min Frame Bit Count Rate Control Enable Mask** Format: U1 Enable Min. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is less than or equal to FrameBitRateMin.

MFX_VP8_Encoder_CFG

Value	Name	Description
1		If (Total Frame Level Bit Counter) =< (Frame Bit Rate Minimum limit) Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS Control Register.
0		Do not update bit[0] of MFX_VP8_IMAGE_STATUS Control Register.

0 — Max Frame bit count Rate Control Enable Mask

Format:	U1

Enable Max. Frame Rate Control. This is a mask bit controlling if the condition of frame level bit count is greater than or equal to FrameBitRateMax.

Value	Name	Description
1		If (Total Frame Level Bit Counter) >= (Frame Bit Rate Maximum Limit) Set bit[0] and bit[1] of MFX_VP8_IMAGE_STATUS control register.
0		Do not update bit[0] of MFX_VP8_IMAGE_STATUS control register.

3 — 31:28 Reserved

27:16 — Max Intra MB Bit Count Limit

Format:	U12

12-bit bit count for Max Intra MB Limit.

15:12 — Reserved

Format:	MBZ

11:0 — Max Inter MB bit count

Format:	U12

12-bit bit count for Max Inter MB Limit.

4 — 31 — Frame Bitrate Min Unit Mode

Format:	U1

This field is the Frame Bitrate Minimum Limit Units.

Value	Name	Description
0h	Compatibility Mode	Frame BitRate Min Unit is in old mode **(128b/16Kb)**
1h	New Mode	Frame BitRate Min Unit is in new mode **(32byte/4Kb)**

30 — Frame Bit Rate Min Unit

Format:	U1

This field is Frame Bitrate Minimum Mode.

Value	Name
0	32-B
1	4-KB

29:16 — Frame Bit Rate Min

Format:	U14

If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values

MFX_VP8_Encoder_CFG

	15	**Frame Bitrate Max Unit Mode** Format: U1 This field is the Frame Bitrate Maximum Limit Units. 0h – Compatibility Mode – Frame BitRate Max Unit is in old mode **(128b/16Kb)** 1h – New Mode – Frame BitRate Max Unit is in new mode **(32byte/4Kb)**
	14	**Frame Bit Rate Max Unit** Format: U1 *This field is Frame Bitrate Maximum Mode* 0 – 32-B 1 – 4-KB
	13:0	**Frame Bit Rate Max** Format: U14 If either BRC Underflow or overflow is enabled. Frame Bit Rate Min and Frame Bit Rate Max need to be programmed with unambiguous values
5	31:24	**Frame Delta QIndex Max[3]** This field is the Frame level delta Qindex for total bit-count above FrameBitRateMax - First 1/8 Region. This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of (FrameBitRateMax, (FrameBitRateMax + FrameBitRateMaxDelta » 3)].
	23:16	**Frame DeltaQ Index Max[2]** This field is the Frame level delta Qindex for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4. This field is used to calculate the suggested Frame Qindex into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3), (FrameBitRateMax+ FrameBitRateMaxDelta » 2)].
	15:8	**Frame Delta QIndex Max[1]** This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/4 and Below 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and 1/2 of FrameBitRateMaxDelta above **FrameBitRateMax**; i.e., In the range of [(FrameBitRateMax+ FrameBitRateMaxDelta » 2), (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].

MFX_VP8_Encoder_CFG		
	7:0	**Frame Delta QIndex Max [0]** This field is the Frame level delta QINDEX for bit-count above FrameBitRateMax - Above 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of **FrameBitRateMax**; i.e., In the range of [(FrameBitRateMax + FrameBitRateMaxDelta » 1), ∞ (Infinite)].
6	31:24	**Frame Delta QIndex Min[3]** This field is the Frame level delta QINDEX for total bit-count below FrameBitRateMin - First 1/8 Region. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 3), FrameBitRateMin].
	23:16	**Frame Delta QIndex Min[2]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin- FrameBitRateMinDelta » 3)].
	15:8	**Frame Delta QIndex Min[1]** This field is the Frame level delta QINDEX for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from **FrameBitRateMin**; i.e., In the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1), (FrameBitRateMin - FrameBitRateMinDelta » 2)].
	7:0	**Frame Delta QIndex Min[0]** This field is the Frame Level Delta QINDEX for bit-count below FrameBitRateMin - Below 1/2. This field is used to calculate the suggested Frame QINDEX into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of **FrameBitRateMin**; i.e., In the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).
7	31:0	**Per Segment Frame Delta QIndex Max[1]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim
8	31:0	**Per Segment Frame Delta QIndex Min[1]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim

MFX_VP8_Encoder_CFG

9	31:0	**Per Segment Frame Delta QIndex Max[2]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim
10	31:0	**Per Segment Frame Delta QIndex Min[2]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim
11	31:0	**Per Segment Frame Delta QIndex Max[3]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim
12	31:0	**Per Segment Frame Delta QIndex Min[3]** Project: CHV, BSW Reserved in CHV, BSW, Need to be programmed with same value as Segment0. Per Segment BRC is not validated in CHV, BSW Fulsim
13	31:24	**Frame Delta Loop Filter Max[3]** Format: U8 This field is the Frame level delta LoopFilter for total bit-count above FrameBitRateMax - First 1/8 region. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame exceeds FrameBitRateMax but is within 1/8 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of (FrameBitRateMax, (FrameBitRateMax+ FrameBitRateMaxDelta » 3)].
	23:16	**Frame Delta Loop Filter Max[2]** Format: U8 This field is the Frame level delta LoopFilter for bit-count above FrameBitRateMax - Above 1/8 and Below 1/4. This field is used to calculate the suggested Frame LoopFilter into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/8 and 1/4 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of ((FrameBitRateMax + FrameBitRateMaxDelta » 3) and (FrameBitRateMax + FrameBitRateMaxDelta » 2)].
	15:8	**Frame Delta Loop Filter Max[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above1/ 4 and Below 1/2.

MFX_VP8_Encoder_CFG

		This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between 1/4 and1/2 of FrameBitRateMaxDelta above FrameBitRateMax. i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 2) and (FrameBitRateMax+ FrameBitRateMaxDelta » 1)].
	7:0	**Frame Delta Loop Filter Max[0]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count above FrameBitRateMax - Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is above FrameBitRateMax by more than half the distance of FrameBitRateMaxDelta. i.e., in the range of ((FrameBitRateMax+ FrameBitRateMaxDelta » 1), infinite).
14	31:24	**Frame Delta Loop Filter Min[3]** Format: U8 This field is the Frame level delta LOOPFILTER for total bit-count below FrameBitRateMin - First 1/8 region. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is less than FrameBitRateMin and greater than or equal to 1/8 the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 3), FrameBitRateMin).
	23:16	**Frame Delta Loop Filter Min[2]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/ 8 and Above 1/4. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between one-eighth and quarter the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 2), (FrameBitRateMin - FrameBitRateMinDelta » 3)).
	15:8	**Frame Delta Loop Filter Min[1]** Format: U8 This field is the Frame level delta LOOPFILTER for bit-count below FrameBitRateMin- Below 1/4 and Above 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is between quarter and half the distance of FrameBitRateMinDelta from FrameBitRateMin. i.e., in the range of [(FrameBitRateMin - FrameBitRateMinDelta » 1) and (FrameBitRateMin - FrameBitRateMinDelta » 2)).

MFX_VP8_Encoder_CFG

	7:0	**Frame Delta Loop Filter Min[0]** Format: U8 This field is the Frame Level Delta LOOPFILTER for bit-count below FrameBitRateMin - Below 1/2. This field is used to calculate the suggested Frame LOOPFILTER into the MFX_VP8_IMAGE_STATUS control register when total bit count for the entire frame is below FrameBitRateMin by more than half the distance of FrameBitRateMinDelta. i.e., in the range of [0, (FrameBitRateMin - FrameBitRateMinDelta » 1).
15	31:0	**Per Segment Frame Delta LoopFilter Max[1]** Project: +, CHV, BSW
16	31:0	**Per Segment Frame Delta LoopFilter Min[1]** Project: +, CHV, BSW
17	31:0	**Per Segment Frame Delta LoopFilter Max[2]** Project: +, CHV, BSW
18	31:0	**Per Segment Frame Delta LoopFilter Min[2]** Project: +, CHV, BSW
19	31:0	**Per Segment Frame Delta LoopFilter Max[3]** Project: +, CHV, BSW
20	31:0	**Per Segment Frame Delta LoopFilter Min[3]** Project: +, CHV, BSW
21	31	**Reserved**
	30:16	**FrameBitRateMinDelta** Format: U15 This field is used to select the frame delta QINDEX when FrameBitRateMin Is exceeded. It shares the same FrameBitrateMinUnit. **Value** / **Name** / **Description** [0-4095] / / When FrameBitrateMinUnit is in Bytes, this range is in Bytes. When FrameBitrateMinUnit is in KB, this range is in KB units.
	15	**Reserved**
	14:0	**Frame Bit Rate Max Delta** Format: U15 This field is used to select the frame delta QINDEX when FrameBitRateMax Is exceeded. It shares the same FrameBitrateMaxUnit. **Value** / **Name** / **Description** [0-4095] / / When FrameBitrateMinUnit is in Bytes, this range is in Bytes. When FrameBitrateMinUnit is in KB, this range is in KB units.
22	31:24	**Reserved**

MFX_VP8_Encoder_CFG

	23	**Show Frame** Format: U1 VP8 Frame Tag, Show Frame Field
	22:20	**Bitstream Format Version** Format: U3 VP8 Frame Tag, Verison Field
	19:18	**Reserved** Format: MBZ
	17:16	**Min Frame WSize Unit** Format: U2 Value / Name / Description: 0h – Compatibility Mode – MinFrameWSizeUnit is in old mode (128b/16Kb) 1h – New Mode – MinFrameWSizeUnit is in new mode (32byte/4Kb)
	15:0	**Min Frame WSize** Exists If: //Encoder Only This field (in Word, 16-bit) is specified to compensate for Intel® Rate Control. Zero padding would be performed.
23	31:16	**Vertical_Size_Code** Format: U16 Frame Tag Vertical Size Code, composed of {VerticalScale[15:14], FrameHeight[13:0]}
	15:0	**Horizontal_Size_Code** Format: U16 Frame Tag Horizontal Size Code, composed of {HorizontalScale[15:14], FrameWidth[13:0]}
24	31:0	**Frame Header Bit Count** Format: U32 Binarized Header Bit Count.
25	31:0	**Frame Header Bin Buffer Qindex Update Pointer** Format: U32 Binarized Header Qindex Update Pointer If Segment Enabled and UpdateSegmentFeature enabled, 4 per segment Qindices would be updated in Binarized header (Only ABS mode supported). Else Base Qindex would be updated
26	31:0	**Frame Header Bin Buffer LoopFilter Update Pointer** Format: U32 Binarized Header LoopFilter Update Pointer If Segment Enabled and UpdateSegmentFeature enabled, 4 per segment LoopFilters would be updated in Binarized header (Only ABS mode supported). Else Base LoopFilter would be updated.

MFX_VP8_Encoder_CFG

27	31:0	**Frame Header Bin Buffer Token Update Pointer** Format: U32 Binarized Header TokenUpdate Pointer
28	31:0	**Frame Header Bin Buffer MVUpdate Pointer** Format: U32 Binarized Header MVUpdate Pointer.
29	31:28	**ClampValues - CV7**
	27:24	**CV6**
	23:20	**CV5**
	19:16	**CV4**
	15:12	**CV3**
	11:8	**CV2**
	7:4	**CV1**
	3:0	**CV0 - Clamp Value 0** Format: U4 If the magnitude of coefficients at locations assigned with CV0 (mapping shown below) exceeds 2CV0-1, they are replaced with 2CV0-1. For coefficients at locations marked as 'none', no clamping is performed. The following mappings are only applied to luma and chroma blocks\subblocks containing AC coefficiencts (blocks\sublocks with only DC coeffs will not be clamped). **For 4x4 frame block, each coefficient is mapped to one of the eight CV values as following:** none CV7 CV5 CV4 CV7 CV6 CV4 CV3 CV5 CV4 CV2 CV1 CV4 CV3 CV1 CV0 **For 4x4 field block, each coefficient is mapped to one of the eight CV values as following:** none CV6 CV3 CV1 CV7 CV6 CV3 CV1 CV5 CV4 CV2 CV0 CV5 CV4 CV2 CV0 Value: 0-15 — Name: **Programming Notes:** In this project, the only value permitted for CV7 through CV0 is 0xf The only value permitted for CV7 through CV0 is 0xf

MFX_VP8_PAK_OBJECT

MFX_VP8_PAK_OBJECT

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MFX_VP8_PAK_OBJECT command is the second primitive command for the VP8 Encoding Pipeline. The MV Data portion of the bitstream is loaded as indirect data object.Before issuing a MFX_VP8_PAK_OBJECT command, all VP8 MFX states need to be valid; therefore the commands used to set these states need to have been issued prior to the issue of this command. MB record must be consecutive with no gaps, hence we do not need MB(x,y) in each MB command. Internal counter will keep track of the current MB address, starting from the first MB. MFX_VP8_PAK_OBJECT command follows the MbType definition like MFD. Encoding statistical data such as the total size of the output bitstream are provided through MMIO registers. Software may access these registers through MI_STORE_REGISTER_MEM command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h MFX_VP8_PAK_OBJECT Format: OpCode
	26:24	**Media Command Opcode** Default Value: 4h VP8_ENC Format: OpCode
	23:21	**SubOpcode A** Default Value: 2h Format: OpCode
	20:16	**SubOpcode B** Default Value: 9h Format: OpCode
	15:12	**Reserved** Project: All Format: MBZ
	11:0	**DWord Length** Default Value: 5h DWORD_COUNT_n Project: All Format: =n Length -2
1	31:30	**Reserved**

MFX_VP8_PAK_OBJECT

	29	**Reserved** Project: CHV, BSW Format: MBZ
	28:10	**Reserved** Project: All Format: MBZ
	9:0	**Indirect PAK-MV Data Length** Format: U10 This field provides the length in bytes of the indirect data, which contains all the MVs for the current MB (in any partitioning and subpartitioning form). A value zero indicates that indirect data fetching is disabled - subsequently, the Indirect PAK-MV Data Start Address field is ignored. This field must have the same alignment as the Indirect PAK-MV Data Start Address. This field must be DW aligned (since each MV is 4 bytes in size).Driver has to derived this field from MVsize (MVquantity in DXVA, exact size) *4 bytes per MV.
2	31:29	**Reserved** Format: MBZ
	28:0	**Indirect PAK-MV Data Start Address Offset** This field specifies the memory starting address (offset) of the MV data to be fetched into PAK Subsystem for processing. This pointer is relative to the MFC Indirect PAK-MV Object Base Address.Hardware ignores this field if indirect data is not present, i.e. the Indirect PAK-MV Data Length is set to 0. It is a Dword aligned address in all AVC encoding configuration, since each MV is 4 bytes in size. **Value** / **Name** [0,512MB)
3..6	127:0	**Inline Data** All the required MB level controls and parameters for encoding are captured as Inline Data Description - VP8 PAK OBJECT. It has a fixed size of 4 DWs. Its definition is described in the next section.

MFX_VP8_PIC_STATE

MFX_VP8_PIC_STATE

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

This must be the very first command to issue after the surface state, the pipe select and base address setting commands and must be issued before MFX_VP8_IMG_STATE.

Programming Notes	Project
Only able to use this instruction for decoder workloads.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Video Codec Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VP8 Format: OpCode
	23:21	**Sub Opcode A** Default Value: 0h VP8 Common Format: OpCode
	20:16	**Sub Opcode B** Default Value: 0h MFX_VP8_PIC_STATE Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Value / Name / Description: 000h — Excludes DWord (0,1) **[Default]** — A special case to provide a dummy image state for stitch mode operation. In this case, fields in DW1 which is part of the dummy image state command are ignored by hardware." 024h — Used for normal decode and encode mode

MFX_VP8_PIC_STATE

1	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**Frame Height Minus 1** Exists If: //Decoder / Encoder Format: U8 Picture Height in integer number of MBs minus 1, so the min pic height can be program is 16 rows of pixels.
	15:8	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	7:0	**Frame Width Minus 1** Exists If: //Decoder / Encoder Format: U8 Picture Width in integer number of MBs minus 1, so the min pic width can be program is 16 pixels.
2	31:26	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	25:24	**Log2 Num of Partition** Exists If: //Decoder / Encoder Format: U2

Value	Name
0	1 Token partition
1	2 Token partition
2	4 Token partition
3	8 Token partition

	23:19	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	18:16	**Deblock Sharpness Level** Exists If: //Decoder / Encoder Format: U3 Specifiy the sharpness level, as one of the regular deblocking strength control parameters.

MFX_VP8_PIC_STATE

		Programming Notes Set to 0 to disable the use of sharpness control.
	15:14	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	13	**Alternate Ref Pic MV SignBias Flag** Exists If: //Decoder / Encoder Alternate Reference Picture MV sign bias flag, specified for non-key frame only.
	12	**Golden Ref Picture MV SignBias Flag** Exists If: //Decoder / Encoder Golden Reference Picture MV sign bias flag, specified for non-key frame only.
	11	**Mode Reference Loop Filter Delta Enabled** Exists If: //Decoder / Encoder Value / Name / Description 0 / / Mode or Reference Loop Filter Delta Adjustment for current frame is disabled. 1 / / Mode or Reference Loop Filter Delta Adjustment for current frame is enabled.
	10	**MB NoCoeff SkipFlag** Exists If: //Decoder / Encoder Frame level control if Skip MB (with no non-zero coefficient) is allowed or not. Value / Name / Description 0 / / All MBs will have its MB level signaling mb_skip_coeff forced to 0. That is, no skip of coefficient record in the bitstream (even their values are all 0s) 1 / / Skip MB is enabled in the per MB record.
	9	**Update MBSegment Map Flag** Exists If: //Decoder / Encoder Value / Name / Description 0 / / Disable segmentation update 1 / / Enable segmentation update, and to enable reading segment_id for each MB.
	8	**Segment Enable Flag** Exists If: //Decoder / Encoder Value / Name / Description 0 / / Disable Segmentation processing in the current frame

MFX_VP8_PIC_STATE

1		Enable Segmentation processing in the current frame

7 Segmentation ID StreamIn Enable

Exists If:	//Decoder Only

Value	Name
0	StreamIn Disabled
1	StreamIn Enabled

Programming Notes
When 0, no input needed.

7:6 Reserved

Exists If:	//Encoder Only
Format:	MBZ

6 Segmentation ID StreamOut Enable

Exists If:	//Decoder Only

Value	Name
0	StreamOut Disabled
1	StreamOut Enabled

Programming Notes
When 0, no output needed.

5 sKeyFrameFlag

Exists If:	//Decoder / Encoder

Value	Name
0	Non-Key Frame (P-Frame)
1	Key Frame (I-Frame)

4 DBLKFilterType

Exists If:	//Decoder / Encoder

To specify VP8 Profile of operation.

Value	Name	Description
0		Use a full feature normal deblocking filter
1		Use a simple filter for deblocking

3:2 Reserved

Exists If:	//Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: MBZ
	1	**Chroma Full Pixel MC Filter Mode** Exists If: //Decoder / Encoder To specify VP8 Profile of operation. Value 0: Chroma MC filter operates in sub-pixel mode Value 1: Chroma MC filter only operates in full pixel position, i.e. no sub-pixel interpolation.
	0	**MC Filter Select** Exists If: //Decoder / Encoder To specify VP8 Profile of operation. Value 0: 6-tap filter (regular filter mode) Value 1: 2-tap bilinear filter (simple profile/version mode)
3	31:30	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	29:24	**DBLKFilterLevel for Segment3** Exists If: //Decoder / Encoder Format: U6 Value 0: Signifies disable in loop deblocking operation — This is used to set a VP8 profile without in loop deblocker. **Programming Notes** There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame. **Restriction** For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128
	23:22	**Reserved** Exists If: //Decoder / Encoder

MFX_VP8_PIC_STATE

	Format:	MBZ

21:16 **DBLKFilterLevel for Segment2**

Exists If:	//Decoder / Encoder
Format:	U6

Value	Name	Description
0	Signifies disable in loop deblocking operation	This is used to set a VP8 profile without in loop deblocker.

Programming Notes
There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.

Restriction
For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128

15:14 **Reserved**

Exists If:	//Decoder / Encoder
Format:	MBZ

13:8 **DBLKFilterLevel for Segment1**

Exists If:	//Decoder / Encoder
Format:	U6

Value	Name	Description
0	Signifies disable in loop deblocking operation	This is used to set a VP8 profile without in loop deblocker.

Programming Notes
There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame.

Restriction
For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128

7:6 **Reserved**

Exists If:	//Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: MBZ
	5:0	**DBLKFilterLevel for Segment0** Exists If: //Decoder / Encoder Format: U6 **Value** / **Name** / **Description** 0 / Signifies disable in loop deblocking operation / This is used to set a VP8 profile without in loop deblocker. **Programming Notes** There are max 4 segments per frame, each segment can have its own deblocking filter level. When segmentation is disabled, only segment 0 parameter is used for the entire frame. **Restriction** / **Project** For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 / CHV, BSW
4 **Project:** CHV, BSW	31	**Reserved** Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	30:24	**Seg 3 Qindex** Project: CHV, BSW Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 3
	23	**Reserved** Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	22:16	**Seg 2 Qindex** Project: CHV, BSW Exists If: //Encoder Only Format: U7 Quantizer Value for Segment ID 2
	15	**Reserved** Project: CHV, BSW

MFX_VP8_PIC_STATE

		Exists If: //Encoder Only
		Format: MBZ
	14:8	**Seg 1 Qindex**
		Project: CHV, BSW
		Exists If: //Encoder Only
		Format: U7
		Quantizer Value for Segment ID 1
	7	**Reserved**
		Project: CHV, BSW
		Exists If: //Encoder Only
		Format: MBZ
	31:0	**Reserved**
		Project: CHV, BSW
		Exists If: //Decoder Only
		Format: MBZ
	6:0	**Seg 0 Qindex**
		Project: CHV, BSW
		Exists If: //Encoder Only
		Format: U7
		Quantizer Value for Segment ID 0.
		Programming Notes
		This is the **[Default]** Qindex
5	31:29	**Reserved**
		Exists If: //Encoder Only
		Format: MBZ
	28	**UVac Qindex Delta Sign**
		Project: CHV, BSW
		Exists If: //Encoder Only
		Format: U1
		Sign of Quantization index delta for UVac
	27:24	**UVac QindexDelta**
		Project: CHV, BSW
		Exists If: //Encoder Only
		Format: U4
		Absolute Quantization index delta for UVac
	23:21	**Reserved**

MFX_VP8_PIC_STATE

		Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	20	**UVdc Qindex Delta Sign** Project: CHV, BSW Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for UVdc
	19:16	**UVdc Qindex Delta** Project: CHV, BSW Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for UVdc
	15:13	**Reserved** Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	12	**Y2ac Qindex Sign** Project: CHV, BSW Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2ac
	11:8	**Y2ac Qindex Delta** Project: CHV, BSW Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2ac
	7:5	**Reserved** Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	4	**Y2ac Qindex Delta Sign**

MFX_VP8_PIC_STATE

		Project: CHV, BSW Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y2dc This is the **[Default]** Qindex Delta Sign
	31:0	**Reserved** Exists If: //Decoder Only Format: MBZ
	3:0	**Y2dc Qindex Delta** Project: CHV, BSW Exists If: //Encoder Only Format: U4 Absolute Quantization index delta for Y2dc This is the **[Default]** Qindex Delta
6	31:5	**Reserved** Exists If: //Encoder Only Format: MBZ
	4	**Y1dc Qindex Delta Sign** Project: CHV, BSW Exists If: //Encoder Only Format: U1 Sign of Quantization index delta for Y1dc This is the **[Default]**Qindex Delta Sign
	31:0	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	3:0	**Y1dc Qindex Delta** Project: CHV, BSW Exists If: //Encoder Only Absolute Quantization index delta for Y1dc This is the **[Default]**Qindex Delta
7	31:15	**Reserved**

MFX_VP8_PIC_STATE

Project: CHV, BSW		Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	14:8	**Clamp Qindex high** Project: CHV, BSW Exists If: //Encoder Only Format: U7 Maximum Clamp Value for Qindex used in quantization.
	7	**Reserved** Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	31:0	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	6:0	**Clamp Qindex Low** Project: CHV, BSW Exists If: //Encoder Only Format: U7 Minimum Clamp Value for Qindex used in quantization.
8	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [1][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved**

MFX_VP8_PIC_STATE

		Exists If: //Encoder Only
		Format: MBZ
	8:0	**Quantizer Value [1][BlockType2=UVDC]**
		Exists If: //Decoder Only
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]
9	31:25	**Reserved**
		Exists If: //Decoder Only
		Format: MBZ
	24:16	**Quantizer Value [1][BlockType5=Y2AC]**
		Exists If: //Decoder Only
		Format: U9
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]
	15:9	**Reserved**
		Exists If: //Decoder Only
		Format: MBZ
	31:0	**Reserved**
		Exists If: //Encoder Only
		Format: MBZ
	8:0	**Quantizer Value [1][BlockType4=Y2DC]**
		Exists If: //Decoder Only
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]
10	31:25	**Reserved**
		Exists If: //Decoder Only
		Format: MBZ
	24:16	**Quantizer Value [2][BlockType1=Y1AC]**
		Exists If: //Decoder Only
		Format: U9
		Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]
	15:9	**Reserved**
		Exists If: //Decoder Only
		Format: MBZ
	31:0	**Reserved**
		Exists If: //Encoder Only
		Format: MBZ
	8:0	**Quantizer Value [2][BlockType0=Y1DC]**

MFX_VP8_PIC_STATE

		Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
11	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [2][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [2][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
12	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [2][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [2][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
13	31:25	**Reserved**

MFX_VP8_PIC_STATE

		Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [3][BlockType1=Y1AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType0=Y1DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
14	31:25	**Reserved** Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [3][BlockType3=UVAC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType2=UVDC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
15	31:25	**Reserved**

MFX_VP8_PIC_STATE

		Exists If: //Decoder Only Format: MBZ
	24:16	**Quantizer Value [3][BlockType5=Y2AC]** Exists If: //Decoder Only Format: U9 **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
	15:9	**Reserved** Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	8:0	**Quantizer Value [3][BlockType4=Y2DC]** Exists If: //Decoder Only **Quantizer Value [n = Segment_Id = 0..3][BlockType = 0..5]**
16	31:6	**CoeffProbability StreamIn Base Address** Exists If: //Decoder Only Format: StreamInAddress[31:6] 64 bytes aligned buffer in linear format. (not tile for better performance) It is specified for non-key frame only. It is the final computed probability table for parsing Coeff in the bitstream. The buffer is unsigned 8-bit * 1056 entries (CoeffProbs[4][8][3][11].
	31:0	**Reserved** Exists If: //Encoder Only Format: MBZ
	5:0	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
17 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	31:0	**Reserved**

MFX_VP8_PIC_STATE

		Project: CHV, BSW Exists If: //Encoder Only Format: MBZ
	15:0	**CoeffProbability StreamIn Address** Project: CHV, BSW Exists If: //Decoder Only This field is for the upper range of CoeffProbability StreamIn Address
18 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	8:7	**CoeffProbability StreamIn - Arbitration Priority Control** Project: CHV, BSW Exists If: //Decoder Only Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

Value	Name
00b	Highest priority
01b	Second highest priority
10b	Third highest priority
11b	Lowest priority

	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for CoeffProbability StreamIn**

MFX_VP8_PIC_STATE

Address

Project:	CHV, BSW
Exists If:	//Decoder Only

This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream.

Value	Name	Description	Exists If
00b	Use Cacheability Controls from page table		//Decoder Only
01b	UC	Uncacheable - non-cacheable	
10b	WT	Writethrough	
11b	WB	Writeback	

4:3 **CoeffProbability StreamIn Address - Target Cache (TC)**

Project:	CHV, BSW
Exists If:	//Decoder Only

This field allows the choice of LLC vs eLLC for caching

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

Project:	CHV, BSW

31:0 **Reserved**

Project:	CHV, BSW
Exists If:	//Encoder Only
Format:	MBZ

1:0 **CoeffProbability StreamIn Address - Age for QUADLRU (AGE)**

Project:	CHV, BSW
Exists If:	//Decoder Only

This field allows the selection of AGE parameter for a given surface in LLC or eLLC.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

MFX_VP8_PIC_STATE

19	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MBSegmentIDTreeProbs[2]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
	15:8	**MBSegmentIDTreeProbs[1]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
	7:0	**MBSegmentIDTreeProbs[0]** Exists If: //Decoder / Encoder Format: U8 MBSegmentIDTreeProbs[2:0] probability tree table for CPBAC parsing Segment_ID of each MB.
20	31:24	**MBNoCoeffSkipFalseProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the MBNoCoeffSkip Flag in the bistream.
	23:16	**IntraMBProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the intra or inter MB type flag in the bitstream.
	15:8	**InterPredFromLastRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
	7:0	**InterPredFromGRefRefProb** Exists If: //Decoder / Encoder Format: U8 8-bit probability value for CPBAC parsing of the flag in the bitstream that determines which reference frame to be used for the current MB motion compensation.
21	31:24	**YModeProb[3]** Exists If: //Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	23:16	**YModeProb[2]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	15:8	**YModeProb[1]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
	7:0	**YModeProb[0]** Exists If: //Decoder / Encoder Format: U8 YModeProb[3:0] probability tree table for CPBAC parsing Luma MBType of each MB.
22	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**UVModeProb[2]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	15:8	**UVModeProb[1]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
	7:0	**UVModeProb[0]** Exists If: //Decoder / Encoder Format: U8 UVModeProb[2:0] probability tree table for CPBAC parsing Chroma MBType of each MB.
23	31:24	**MVUpdateProbs[0][3]** Exists If: //Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][2]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][1]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][0]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
24	31:24	**MVUpdateProbs[0][7]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][6]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][5]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][4]** Exists If: //Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
25	31:24	**MVUpdateProbs[0][11]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	23:16	**MVUpdateProbs[0][10]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	15:8	**MVUpdateProbs[0][9]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
	7:0	**MVUpdateProbs[0][8]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB.
26	31:24	**MVUpdateProbs[0][15]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[0][14]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][13]** Exists If: //Decoder / Encoder

MFX_VP8_PIC_STATE

		Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][12]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
27	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MVUpdateProbs[0][18]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[0][17]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[0][16]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
28	31:24	**MVUpdateProbs[1][3]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][2]**

MFX_VP8_PIC_STATE

		Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][1]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][0]** Exists If: //Decoder Only Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
29	31:24	**MVUpdateProbs[1][7]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][6]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][5]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][4]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
30	31:24	**MVUpdateProbs[1][11]**

MFX_VP8_PIC_STATE

		Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][10]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][9]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][8]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
31	31:24	**MVUpdateProbs[1][15]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	23:16	**MVUpdateProbs[1][14]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][13]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][12]**

MFX_VP8_PIC_STATE

		Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
32	31:24	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	23:16	**MVUpdateProbs[1][18]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	15:8	**MVUpdateProbs[1][17]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
	7:0	**MVUpdateProbs[1][16]** Exists If: //Decoder / Encoder Format: U8 MVUpdateProbs[1:0][18:0] probability table for CPBAC parsing of MV update value of each MB. To map into DWord, it becomes MVUpdate[1:0][19:0].
33	31	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	30:24	**RefLFDelta3 (for ALTREF FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.

MFX_VP8_PIC_STATE

		Restriction For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128
	23	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	22:16	**RefLFDelta2 (for GOLDEN FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude. **Restriction** / **Project** For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 / CHV, BSW
	15	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	14:8	**RefLFDelta1 (for LAST FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude. **Restriction** / **Project**

MFX_VP8_PIC_STATE

		For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 — CHV, BSW
	7	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	6:0	**RefLFDelta0 (for INTRA FRAME)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for reference frame based adjustment of the MB-level's filter level value. RefLFDeltas [ref_frametype = 0 to 3] **Programming Notes** Please note that although RefDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude. **Restriction** / **Project** For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 — CHV, BSW
34	31	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	30:24	**ModeLFDelta3 (for SPLITMV mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude. **Restriction** / **Project**

MFX_VP8_PIC_STATE

		For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128	CHV, BSW
	23	**Reserved** Exists If: //Decoder / Encoder Format: MBZ	
	22:16	**ModeLFDelta2 (for Nearest, Near and New mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.	
		Restriction	**Project**
		For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128	CHV, BSW
	15	**Reserved** Exists If: //Decoder / Encoder Format: MBZ	
	14:8	**ModeLFDelta1(for ZEROMV mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude.	
		Restriction	**Project**

MFX_VP8_PIC_STATE

		For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 — CHV, BSW
	7	**Reserved** Exists If: //Decoder / Encoder Format: MBZ
	6:0	**ModeLFDelta0 (for B_PRED mode)** Exists If: //Decoder / Encoder Format: S6 2's Compliment Delta value for mode based adjustment of the MB-level's filter level value. ModeLFDeltas[MB_Type = 0 to 3] **Programming Notes** Please note that although ModeLFDelta is signed 2's complement, bitstream is sign bit + 6 bit magnitude. **Restriction** / **Project** For VP8 Encoder: Max (DBLKFilterLevel for Segment0, DBLKFilterLevel for Segment1, DBLKFilterLevel for Segment2, DBLKFilterLevel for Segment3) + Max (RefLFDelta0, RefLFDelta1, RefLFDelta2, RefLFDelta3) + Max (ModeLFDelta0, ModeLFDelta1, ModeLFDelta2, ModeLFDelta3) < 128 — CHV, BSW
35 **Project:** CHV, BSW	31:0	**Segmentation ID Stream Base Address** Project: CHV, BSW Exists If: //Decoder Only Format: StreamAddress[31:0] 64 bytes linear aligned buffer It is specified when **SegmentationIDStreamInEnable** or **SegmentationIDStreamOutEnable** is specified. **Programming Notes** Each cache has only 8 bits for 4 segmentation ID from 4 continuous MBs.
36 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	15:0	**Segmentation ID Stream Base Address [47:32]**

MFX_VP8_PIC_STATE

		Project: CHV, BSW Exists If: //Decoder Only This field is for the upper range of **Segmentation ID Stream Base Address**
37 **Project:** CHV, BSW	31:15	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	14:13	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	12:11	**Reserved** Project: CHV, BSW Exists If: //Decoder Only Format: MBZ
	10:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Segmentation ID Stream - Arbitration Priority Control** Project: CHV, BSW Exists If: //Decoder Only Format: U2 This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. **Value** / **Name** 00b – Highest priority 01b – Second highest priority 10b – Third highest priority 11b – Lowest priority
	6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC) for Segmentation ID Stream Base Address** Project: CHV, BSW Exists If: //Decoder Only This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. **Value** / **Name** / **Description**

MFX_VP8_PIC_STATE

00b	Use Cacheability Controls from page table	
01b	UC	Uncacheable - non-cacheable
10b	WT	Writethrough
11b	WB	Writeback

4:3 **Target Cache (TC) Segmentation ID Stream Base Address**

Project:	CHV, BSW
Exists If:	//Decoder Only

This field controls the L3$, LLC and eLLC (eDRAM) cacheability for a given surface. Setting of "00" points to PTE settings which defaults to eDRAM (when present). If no eDRAM, the access will be allocated to LLC. Setting of "01", allocates into LLC and victimizes the line to eDRAM. Setting of "10" allows the line to be allocated in either LLC or eDRAM. Setting of "11" is the only option for a memory access to be allocated in L3$ as well as LLC/eLLC
00b: eLLC Only *("00" setting points TC selection to PTE which defaults to eLLC)*
01b: LLC Only (W*orks at the allocation time, later victimization from LLC downgrades the line to eLLC if present*).
10b: LLC/eLLC Allowed.
11b: L3, LLC, eLLC Allowed.
Errata CHV:A-E (FIXED BY:G0 Stepping):
For all system that does NOT use SVM (i.e. coherent L3$ surfaces), back snoops from LLC has to be disabled (**Dis_GtCvUpdtOnRd = "1"**). Than target Cache settings can be programmed as POR requirements of L3/LLC/eDRAM caching.
For all systems that does use SVM (i.e. coherent L3$ surfaces), the recomended setting would be "00" in target cache settings. In case of L3 surfaces, the performance has to be tuned between "00" and "11" setting based on the benefits of L3 caching outweighting the degradation of backsnoops.
Post G0-stepping, the above w/a for coherent L3$ surfaces is not needed.

Value	Name
00b	eLLC Only - not snooped in GT
01b	LLC Only
10b	LLC/eLLC Allowed
11b	L3, LLC, eLLC Allowed

2 **Reserved**

1:0 **Age for QUADLRU (AGE) Segmentation ID Stream Base Address**

Project:	CHV, BSW
Exists If:	//Decoder Only

This field allows the selection of AGE parameter for a given surface in LLC or eLLC. If a particular allocation is done at youngest age ("0,1,2") it tends to stay longer in the cache. This option is given to GFX software to be able to decide which surfaces are more likely to generate

MFX_VP8_PIC_STATE

HITs, hence need to be replaced least often in caches.

Value	Name
11b	Good chance of generating hits.
10b	Next good chance of generating hits
01b	Decent chance of generating hits
00b	Poor chance of generating hits

MFX_WAIT

MFX_WAIT

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	1

This command can be considered the same as an MI_NOOP except that the command parser will not parse the next command until the following happens

- **AVC or VC1 BSD mode:** The command will stall the parser until completion of the BSD object
- **IT, encoder, and MPEG2 BSD mode:** The command will stall the parser until the object package is sent down the pipelineThis command should be used to ensure the preemption enable window occurs during the time the object command is being executed down the pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 03h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Command Subtype** Default Value: 01h MFX_SINGLE_DW Format: OpCode
	26:16	**Sub-Opcode** Default Value: 0h MFX_WAIT Format: OpCode
	15:10	**Reserved** Project: All Format: MBZ
	9	**Reserved**
	8	**MFX Sync Control Flag** If set, VCS will stall the parser until all prior MFX objects are completed down the MFX pipeline
	7:6	**Reserved** Project: All Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2

MI_ARB_CHECK

MI_ARB_CHECK

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 1

Description

The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.

Programming Notes

This instruction cannot be placed in a batch buffer.

If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.

DWord	Bit	Description
0	31:29	**MI Instruction Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Instruction Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Project: CHV, BSW
Source: BlitterCS
Length Bias: 1

Description	Project
The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.	

Programming Notes	Project
This instruction cannot be placed in a batch buffer.	
If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Project: CHV, BSW

Source: RenderCS

Length Bias: 1

Description

The MI_ARB_CHECK instruction is used to check the ring buffer double buffered head pointer (register UHPTR). This instruction can be used to pre-empt the current execution of the ring buffer. Note that the valid bit in the updated head pointer register needs to be set for the command streamer to be pre-empted.

Programming Notes

Ring Buffer mode of scheduling:

- The current head pointer is loaded with the updated head pointer register independent of the location of the updated head.
- If the current head pointer and the updated head pointer register are equal, hardware will automatically reset the valid bit corresponding to the UHPTR.
- For pre-emption, the wrap count in the ring buffer head register is no longer maintained by hardware. The hardware updates the wrap count to the value in the UHPTR register.

Execlist mode of scheduling:
MI_ARB_CHECK will be used to indicate a command boundary on which Preemption will be honored by Command Streamer in the execlist mode of operation. UHPTR is ignored when processing MI_ARB_CHECK in execlist mode.

This instruction can be in either a ring buffer or batch buffer.

MI_ARB_CHECK command must not be programmed in INDIRECT_CTX and BB_PER_CTX_PTR buffers.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_CHECK

MI_ARB_CHECK

Project: CHV, BSW
Source: VideoCS
Length Bias: 1

Description

The MI_ARB_CHECK is used to check for a change in arbitration. If executed as part of a Ring Buffer the command checks the UHPTR valid bit and if set the head of the ring will jump to the value of the head pointer programmed in the UHPTR.

Programming Notes

This instruction cannot be placed in a batch buffer.

If execlist is enabled, there is a pending execution list and this command is parsed, then the command streamer will preempt the current context and start executing the new execution list.

DWord	Bit	Description
0	31:29	**MI Instruction Type** Default Value: 0h MI_INSTRUCTION Format: OpCode
	28:23	**MI Instruction Opcode** Default Value: 05h MI_ARB_CHECK Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_ARB_ON_OFF

MI_ARB_ON_OFF

Project:	CHV, BSW
Source:	CommandStreamer
Length Bias:	1

The MI_ARB_ON_OFF instruction is used to disable/enable context switching. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.) This command should always be used as an off-on pair with the sequence of instructions to be protected from context switch between MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

Execution List Mode of Scheduling: The MI_ARB_ON_OFF instruction is used to disable/enable context switching. Context swiching could be either due to preemption or un-succesfull wait for events or semaphore waits. This instruction can be used to prevent submission of a new execlist from interrupting a command sequence, however lite restore preemption is allowed with in the arbitration disabled command execution zone. Note that context switching will remain disabled until re-enabled through use of this command. This command will also prevent a switch in the case of waiting on events, running out of commands. These will effectively hang the device if allowed to occur while arbitration is off (context switching is disabled.)
Ring Buffer Mode of Scheduling: The MI_ARB_ON_OFF instruction is used to disable preemption on the preemptable commands. SW can explicitly make section of commands in a command buffer non-preemptable by sandwitching them between ARB_OFF and ARB_ON, HW will ingore preemption request (UHPTR Valid) until arbitration is enabled.

Programming Notes

This command must be always be programmed in pairs of off/on in the same command dispatch. Sequence of instructions to be protected from cntext switch or preemption must be programmed between the MI_ARB_OFF and MI_ARB_ON. Software must use this arbitration control with caution since it has the potential to increase the response time of the Render Engine to pre-emption requests. This is a privileged command; it will not be effective (will be converted to a no-op) if executed from within a non-privileged batch buffer.

HW doesn't treat Arbitration Disabled as equivalent to "Inhibit Synchronous Context Switch" set in CTXT_SR_CTL register. Power management optimizations (RDOP on WT4EVT) available on setting "Inhibit Synchronous Context Switch" are not enabled by default on Arbitration Disabled. SW must explicitly program "Inhibit Synchronous Switch" when Arbitration Disabled to enable power management optimizations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode

MI_ARB_ON_OFF		
	28:23	**MI Command Opcode** Default Value: 08h MI_ARB_ON_OFF Format: OpCode
	22:1	**Reserved** Format: MBZ
	0	**Arbitration Enable** Format: Enable This field enables or disables context switches due to pre-emption (a new execlist).

MI_ATOMIC

MI_ATOMIC

Project: CHV, BSW
Source: PRM
Length Bias: 2

Description

MI_ATOMIC is used to carry atomic operation on data in graphics memory. Atomic operations are supported on data granularity of 4B, 8B and 16B. The atomic operation leads to a read-modify-write operation on the data in graphics memory with the option of returning value. The data in graphics memory is modified by doing arithmetic and logical operation with the inline/indirect data provided with the MI_ATOMIC command. Inline/Indirect provided in the command can be one or two operands based on the atomic operation. Ex: Atomic-Compare operation needs two operands while Atomic-Add operation needs single operand and Atomic-increment requires no operand. Refer Vol1i L3 URB [CHV, BSW] B-spec for detailed atomic operations supported. Atomic operations can be enabled to return value by setting "Return Data Control" field in the command, return data is stored to CS_GPR registers.
CS_GPR4/5 registers are updated with memory Return Data based on the "Data Size". Each GPR register is qword in size and occupies two MMIO registers.
Note: Any references to CS_GPR registers in the command should be understood as the CS_GPR registers belonging to the corresponding engines *CS_GPR registers.

Engine Name	Corresponding GPR Registers
RCS	CS_GPR
BCS	BCS_GPR
VCS	VCS_GPR
VECS	VECS_GPR

Indirect Source Operands:
Operand1 is sourced from [CS_GPR1, CS_GPR0]
Operand2 is sourced from [CS_GPR3, CS_GPR2]
Read return Data is stored in [CS_GPR_5, CS_GPR4]

When "Data Size" is QWORD or DWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory.

Programming Notes

- When Inline Data mode is not set, Dwords 3..10 must not be included as part of the command. Dword Length field in the header must be programmed accordingly.
- When Inline Data Mode is set, Dwords3..10 must be included based on the Data Size field of the header. Both Operand-1 and Operand-2 dwords must be programmed based on the Data Size field. Operand-2 must be programmed to 0x0 if the atomic operation doesn't require it. Dword Length field in the header must be programmed accordingly.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td>Command Type
<table>
<tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>28:23</td><td>MI Command Opcode
<table>
<tr><td>Default Value:</td><td>2Fh MI_ATOMIC</td></tr>
<tr><td>Format:</td><td>OpCode</td></tr>
</table></td></tr>
<tr><td></td><td>22</td><td>Memory Type
This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>Per Process Graphics Address</td><td></td></tr>
<tr><td>1h</td><td>Global Graphics Address</td><td>This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.</td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td>Reserved
<table>
<tr><td>Source:</td><td>BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>21</td><td>Post-Sync Operation
<table>
<tr><td>Source:</td><td>RenderCS</td></tr>
</table>
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0h</td><td>No Post Sync Operation</td><td>Command is executed as usual.</td></tr>
<tr><td>1h</td><td>Post Sync Operation</td><td>MI_ATOMIC command is executed as a pipelined PIPE_CONTROL flush command with Atomics operation as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding flushes issued prior to this command.
When this bit set following ristiriciton apply to atomic operation:
Non-Compare atomic operations are supported on data granularity of 4B and 8B. DW3 is the lower dword of the operand and DW4 is the upper dword of the operand for the atomic operation.
Compare atomic operations are supported on data granularity of 4B. DW3 is Operand-0 and DW4 is Operand-1 for the atomic operation.
Atomic operations to GGTT/PPGTT memory surface are supported.</td></tr>
</table></td></tr>
</table>

MI_ATOMIC

		• Only Inline data mode for atomic operand is supported, no support for indirect data mode. • No support for Return Data Control functionality. • No support for atomic operations on data granularity of 16B. • No support for compare atomic operations on data granularity of 8B.

Programming Notes
Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command.
AWhen this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued.
AWhen this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.
This bit must not be set due to known HW issues. OR "Post Sync Operation" feature in MI_ATOMIC can be enabled provided "CS Stall" bit is always set in all PIPE_CONTROL commands programmed with "Post Sync Operation" set to "Report Time Stamp" or "Report PS Depth Count".

Workaround
Workaround: "Post Sync Operation" bit must not be set when MI_ATOMIC command is programmed by GPGPU and MEDIA workloads (i.e when PIPELINE_SELECT command is set to GPGPU or MEDIA). This is to WA FFDOP CG issue.

20:19 **Data Size**

This field indicates the size of the operand in dword/qword/octword on which atomic operation will be performed. Data size must match with the Atomic Opcode. Operation Data size could be 4B, 8B or 16B

Value	Name	Description
0h	DWORD	Operand size used by Atomic Operation is DWORD.
1h	QWORD	Operand Size used by Atomic Operation is QWORD.
2h	OCTWORD	Operand Size used by Atomic Operation is OCTWORD.
3h	RESERVED	

18 **Inline Data**

This bit when set indicates the source operands are provided in line within the command. When reset the source operands are in CS_GPR registers.

Programming Notes

MI_ATOMIC

		CS_GPR registers must be programmed with appropriate values before issuing MI_ATOMIC command with this field reset.
	17	**CS STALL** This bit when set command stream waits for completion of this command before executing the next command. **Programming Notes** / **Source**: Render Command Streamer Only: CS will not guarantee atomic operation to be complete upon setting this bit along with Post Sync Operation set. When Post Sync Operation is set, this bit has no significance. — RenderCS **Workaround**: Workaround When CS STALL bit is set, Return Data Control must also be set in MI_ATOMIC command.
	16	**Return Data Control** Project: CHV, BSW Source: RenderCS, BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS **Description**: When "Data Size" is QWORD or DWORD only CS_GPR4 (Qword) is updated with the qword data returned from memory. When the data size is OCTWORD CS_GPR4/5 are updated with the OCTWORD data returned from memory. CS_GPR4 is loaded with lower qword returned from memory and CS_GPR5 is loaded with upper qword returned from memory **Workaround**: Workaround: When Return Data Control bit is set, CS STALL must also be set in MI_ATOMIC command.
	15:8	**ATOMIC OPCODE** This field selects the kind of atomic operation to be performed. Refer Vol1i L3 URB [CHV, BSW] B-spec for atomic opcode corresponding to an atomic operation. **Programming Notes**: Atomic Opcode must not be set to 0x00 (no-atomic).
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). **Value** / **Name** / **Exists If**: 1h / **[Default]** / ([Inline Data]==0); 9h / / ([Inline Data]==1)
1	31:2	**Memory Address**

MI_ATOMIC

		Project: All Format: GraphicsAddress[31:2] This field contains the graphics memory address of the data on which atomic operation has to be performed. Atomic operation can be performed on data granularity of 4B, 8B or 16B and hence the Address has to be correspondingly aligned to 4B,8B or 16B respectively. **Programming Notes** / **Project** Memory Address must be qword aligned for all dword atomic operations. Upper Dword of the memory location should be initialized to 0x0. / CHV, BSW
	1:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Memory Address High** This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
3	31:0	**Operand1 Data Dword 0** Format: U32 Dword0 of Operand1 when Inline Data mode is set.
4	31:0	**Operand2 Data Dword 0** Format: U32 Dword0 of Operand2 when Inline Data mode is set.
5	31:0	**Operand1 Data Dword 1** Format: U32 Dword1 of Operand1 when Inline Data mode is set.
6	31:0	**Operand2 Data Dword 1** Format: U32 Dword1 of Operand2 when Inline Data mode is set.
7	31:0	**Operand1 Data Dword 2** Format: U32 Dword2 of Operand1 when Inline Data mode is set.
8	31:0	**Operand2 Data Dword 2** Format: U32 Dword2 of Operand2 when Inline Data mode is set.

MI_ATOMIC		
9	31:0	**Operand1 Data Dword 3** Format: U32 Dword3 of Operand1 when Inline Data mode is set.
10	31:0	**Operand2 Data Dword 3** Format: U32 Dword3 of Operand2 when Inline Data mode is set.

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END

Project: CHV, BSW
Source: VideoEnhancementCS
Length Bias: 1

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_BATCH+_BUFFER_END Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Project:	CHV, BSW	
Source:	BlitterCS	
Length Bias:	1	
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END
	22:0	**Reserved** Project: All Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_ BATCH_BUFFER_END Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_END

MI_BATCH_BUFFER_END		
Project:	CHV, BSW	
Source:	VideoCS	
Length Bias:	1	
The MI_BATCH_BUFFER_END command is used to terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ah MI_BATCH+_BUFFER_END Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch* buffer. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*. The batch buffer can be specified as privileged or non-privileged, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

Programming Notes

- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.
- It is essential that the address location beyond the current page be populated inside the GTT. HW performs over-fetch of the command addresses and any over-fetch requires a valid TLB entry. A single extra page beyond the batch buffer is sufficient.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**2nd Level Batch Buffer** The command streamer contains three storage elements; one for the ring head address, one for the batch head address, and one for the 2nd level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the 1st level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the 1st (traditional) level batch buffer address. this allows hardware to mimic a simple 3-level stack.

Value	Name	Description
0h	1st level batch	Place the batch buffer address in the 1st (traditional) level batch address storage element.
1h	2nd level batch	Place the batch buffer address in the 2nd-level batch address storage element.

MI_BATCH_BUFFER_START

		Programming Notes Within a second level batch buffer there can't be any chained batch buffers. MI_BATCH_BUFFER_START command is not allowed inside a second level batch buffer.
	21:17	**Reserved** Format: MBZ
	16	**Add Offset Enable** Format: Enable If this bit is set then the value stored in the BB_OFFSET MMIO register will be added to the Batch Buffer Start Address and the summation will be used as the address to fetch from memory. **Specific to the render command stream only.**
	15	**Predication Enable** This bit is used to enable predication of this command. If this bit is set and Bit 0 of the Predicate Result-1 register is clear, this command is ignored. Otherwise the command is performed normally. **Specific to the Render command stream only.**
	14:12	**Reserved**
	11	**Reserved**
	10	**Resource Streamer Enable** Format: Enable When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer. **Specific to the Render command stream only.**
	9	**Reserved**
	8	**Address Space Indicator** Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain. • Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT. • Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.

Value	Name	Description
0h	GGTT	This batch buffer is located in GGTT memory and is privileged.
1h	PPGTT	This batch buffer is located in PPGTT memory and is Non-Privileged.

Programming Notes
This field must be '0' unless the Per-Process GTT Enable is '1'

MI_BATCH_BUFFER_START

	7:0	**DWord Length** Default Value: 1h Format: =n Total - Bias. Excludes DWord (0,1).
1	31:2	**Batch Buffer Start Address** Format: GraphicsAddress[31:2]BatchBuffer This field specifies Bits 31:2 of the starting address of the batch buffer.
	1:0	**Reserved** Format: MBZ
2	31:16	**Reserved**
	15:0	**Batch Buffer Start Address High** Format: GraphicsAddress[47:32]BatchBuffer This field specifies the 4GB aligned base address of Gfx 4GB virtual address spece within the hosts 64-bit virtual address space.

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Project: CHV, BSW
Source: VideoEnhancementCS
Length Bias: 2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a *batch buffer*. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of *MI Functions*.

The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of *MI Functions*.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**2nd Level Batch Buffer** The command streamer contains 3 storage elements; 1 for the ring head address, 1 for the batch head address, and 1 for the 2nd level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the 1st level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the 1st (traditional) level batch buffer address. this allows hardware to mimic a simple 3 level stack. Value / Name / Description 0h / 1st level batch / Place the batch buffer address in the 1st (traditional) level batch address storage element 1h / 2nd level batch / Place the batch buffer address in the 2nd level batch address storage element
	21:13	**Reserved** Format: MBZ
	12	**Reserved**
	11:9	**Reserved** Format: MBZ
	8	**Address Space Indicator** Project: CHV, BSW Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g.,

MI_BATCH_BUFFER_START

MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain.

- Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT.
- Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.

Value	Name	Description
0h	GGTT	This batch buffer is located in GGTT memory and is privileged.
1h	PPGTT	This batch buffer is located in PPGTT memory and is Non-Privileged.

Programming Notes
This field must be '0' unless the Per-Process GTT Enable is '1'

7:0 **DWord Length (Excludes D-Word 0,1) = 0**

Value	Name	Project
1h	Excludes DWord (0,1) **[Default]**	CHV, BSW

1 — 31:2 **Batch Buffer Start Address**

Format:	GraphicsAddress[31:2]

Programming Notes
• A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command. • The selection of PPGTT vs. GGTT for the batch buffer is determined by the **Buffer Security Indicator** (bit 8).

1:0 **Reserved**

Format:	MBZ

2 **Project:** CHV, BSW — 31:16 **Reserved**

Project:	CHV, BSW
Format:	MBZ

15:0 **Batch Buffer Start Address High**

Project:	CHV, BSW
Format:	GraphicsAddress[47:32]BatchBuffer

This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a batch buffer. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of MI Functions.The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of MI Functions.

Programming Notes

- Batch buffers referenced with physical addresses must not extend beyond the end of the starting physical page (can't span physical pages). However, a batch buffer initiated using a physical address can chain to another buffer in another physical page.
- A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**2nd Level Batch Buffer** Project: CHV, BSW The command streamer contains 3 storage elements; 1 for the ring head address, 1 for the batch head address, and 1 for the 2nd level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the 1st level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the 1st (traditional) level batch buffer address. this allows hardware to mimic a simple 3 level stack. Value / Name / Description 0h / 1st level batch / Place the batch buffer address in the 1st (traditional) level batch address storage element 1h / 2nd level batch / Place the batch buffer address in the 2nd level batch address storage element
	21:9	**Reserved**

MI_BATCH_BUFFER_START

	8	**Address Space Indicator** Project: CHV, BSW Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain. • Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT. • Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.
		Value / Name / Description 0h / GGTT / This batch buffer is located in GGTT memory and is privileged. 1h / PPGTT / This batch buffer is located in PPGTT memory and is Non-Privileged.
		Programming Notes This field must be '0' unless the Per-Process GTT Enable is '1'
	7:0	**DWord Length** Format: =n Total - Bias
		Value / Name / Project 1h / Excludes DWord (0,1) **[Default]** / CHV, BSW
1	31:2	**Batch Buffer Start Address** Format: GraphicsAddress[31:2]BatchBuffer This field specifies Bits 31:2 of the starting address of the batch buffer.
	1:0	**Reserved** Format: MBZ
2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Batch Buffer Start Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]BatchBuffer This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_BATCH_BUFFER_START

MI_BATCH_BUFFER_START

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MI_BATCH_BUFFER_START command is used to initiate the execution of commands stored in a batch buffer. For restrictions on the location of batch buffers, see Batch Buffers in the Device Programming Interface chapter of MI Functions.The batch buffer can be specified as secure or non-secure, determining the operations considered valid when initiated from within the buffer and any attached (chained) batch buffers. See Batch Buffer Protection in the Device Programming Interface chapter of MI Functions.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 31h MI_BATCH_BUFFER_START Format: OpCode
	22	**2nd Level Batch Buffer** Project: CHV, BSW The command streamer contains 3 storage elements; 1 for the ring head address, 1 for the batch head address, and 1 for the 2nd level batch head address. When performing batch buffer chaining, hardware simply updates the head pointer of the 1st level batch address storage. There is no stack in hardware. When this bit is set, hardware uses the 2nd level batch head address storage element. Upon MI_BATCH_BUFFER_END, it will automatically return to the 1st (traditional) level batch buffer address. this allows hardware to mimic a simple 3 level stack. Value / Name / Description: 0h / 1st level batch / Place the batch buffer address in the 1st (traditional) level batch address storage element 1h / 2nd level batch / Place the batch buffer address in the 2nd level batch address storage element **Programming Notes** • 2nd level batch buffer chaining is not supported.
	21:10	**Reserved**
	9	**Reserved**

MI_BATCH_BUFFER_START

	8	**Address Space Indicator** Batch buffers accessed via PPGTT are considered as non-privileged. Certain operations (e.g., MI_STORE_DATA_IMM commands to GGTT memory) are prohibited within non-privileged buffers. More details mentioned in User Mode Privileged command section. When MI_BATCH_BUFFER_START command is executed from within a batch buffer (i.e., is a "chained" or "second level" batch buffer command), the current active batch buffer's "Address Space Indicator" and this field determine the "Address Space Indicator" of the next buffer in the chain. • Chained or Second level batch buffer can be in GGTT or PPGTT if the parent batch buffer is in GGTT. • Chained or Second level batch buffer can only be in PPGTT if the parent batch buffer is in PPGTT. This is enforced by Hardware.

Value	Name	Description
0h	GGTT	This batch buffer is located in GGTT memory and is privileged.
1h	PPGTT	This batch buffer is located in PPGTT memory and is Non-Privileged.

Programming Notes
This field must be '0' unless the Per-Process GTT Enable is '1'.

	7:0	**DWord Length** Format: =n Total Length - 2

Value	Name	Project
1h	Excludes DWord (0,1) **[Default]**	CHV, BSW

1	31:2	**Batch Buffer Start Address** Format: GraphicsAddress[31:2]

Programming Notes
• A batch buffer initiated with this command must end either with a MI_BATCH_BUFFER_END command or by chaining to another batch buffer with an MI_BATCH_BUFFER_START command. • The selection of PPGTT vs. GGTT for the batch buffer is determined by the Buffer Security Indicator (bit8).

	1:0	**Reserved**
2 **Project:** CHV, BSW	31:16	**Reserved**
	15:0	**Batch Buffer Start Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]BatchBuffer This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_CLFLUSH

MI_CLFLUSH

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Flushes out the page given in the command out to system memory. This command is specific to the render engine and is not privileged.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 27h Store DW MI_CLFLUSH Format: OpCode
	22	**Use Global GTT** 0h – Per Process Graphics Address 1h – Global Graphics Address: This command will use the global GTT to translate the Address.
	21:10	**Reserved** Format: MBZ
	9:0	**DWord Length** Format: n Total Length - 2 1h – **[Default]** – Excludes DWord (0,1) Programming Notes: The value of this field must not exceed a value 3Fh when programmed in a batch buffer with resource streamer enabled. (Project: CHV, BSW)
1	31:12	**Page Base Address** Format: GraphicsAddress[31:12] 4KB aligned Page Address which software requires hardware to flush to DRAM.
	11:6	**Starting Cacheline Offset** Format: U6 Zero based starting cacheline offset in to the Page Base Address

MI_CLFLUSH

	5:0	**Reserved** Format: MBZ
2	31:16	**Reserved** Format: MBZ
	15:0	**Page Base Address High** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
3..n	31:0	**DW Representing a Half Cache Line** Format: MBZ The information given to hardware is the DW itself, not the contents. Hardware uses the DW count of the command to determine the offset from the base to flush out. The offset is ½ cache line (8 DW = 1HW) granular so for a full page, the command will need 4096 bytes / 4 bytes per DW / 8 DW per HW = 128 DW. **Programming Notes** Always even number of "DW Representing 1/2 cacheline" terms must be programmed.

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command. Termination of second level batch buffer due to this command will also terminate the parent/first level batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Default Value: 0h Format: Boolean If set, this command uses the global GTT to translate the Compare Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT is used to translate the Compare Address.
	21	**Compare Semaphore** Default Value: 0h Format: Boolean If set, the value from the Compare Data Dword is compared to the value from the Compare Address in memory. If the value at Compare Address is greater than the Compare Data Dword, execution of the current command buffer should continue. If clear, the parser will continue to the next command and not exit the batch buffer.
	20	**Reserved**
	19:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**Compare Data Dword** Data DWord to compare to memory. The Data DWord is supplied by software to control

MI_CONDITIONAL_BATCH_BUFFER_END

		execution of the command buffer. If the compare is enabled and the data at Semaphore Address is greater than this DWord, the execution of the command buffer should continue.
2	31:3	**Compare Address** Format: GraphicsAddress[31:3] Qword address to fetch Data Dword(DW0) from memory. HW will compare the Data Dword(DW0) with Compare Data Dword
	2:0	**Reserved** Project: All Format: MBZ
3	31:16	**Reserved** Format: MBZ
	15:0	**Compare Address High** Format: GraphicsAddress[47:32] This field specifies the 4 GB-aligned base address of GFX 4 GB virtual address space within the host's 64-bit virtual address space.

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 2

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command. Termination of second level batch buffer due to this command will also terminate the parent/first level batch buffer.

Programming Notes

This command is only valid with a 1st level batch buffer (bit 22 in MI_BATCH_BUFFER_START is set to '0')

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Default Value: 0h Format: Boolean If set, this command will use the global GTT to translate the **Compare Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Compare Address**.
	21	**Compare Semaphore** Default Value: 0h Format: Boolean If set, the value from the **Compare Data Dword** is compared to the value from the **Compare Address** in memory. If the value at **Compare Address is greater than the Compare Data Dword**, execution of current command buffer should continue. If clear, the parser will continue to the next command and not exit the batch buffer.
	20	**Reserved**
	19:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2

MI_CONDITIONAL_BATCH_BUFFER_END

		Value	Name	Project
		1h	Excludes DWord (0,1) **[Default]**	CHV, BSW
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at **Semaphore Address** is greater than this dword, the execution of the command buffer should continue.		
2	31:3	**Compare Address** Format: GraphicsAddress[31:3] Qword address to fetch Data Dword(DW0) from memory. HW will compare the Data Dword(DW0) with Compare Data Dword		
	2:0	**Reserved** Format: MBZ		
3 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ		
	15:0	**Compare Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space		

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command. Termination of second level batch buffer due to this command will also terminate the parent/first level batch buffer.

Programming Notes

This command is only valid with a 1st level batch buffer (bit 22 in MI_BATCH_BUFFER_START is set to 0).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Default Value: 0h DefaultVaueDesc Format: Boolean Format: U1 FormatDesc **Description** / **Project**: If set, this command will use the global GTT to translate the Compare Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the Compare Address. / CHV, BSW
	21	**Compare Semaphore** Default Value: 0h DefaultVaueDesc Format: Boolean If set, the value from the Compare Data Dword is compared to the value from the Compare Address in memory. If the value at Compare Address is greater than the Compare Data Dword, execution of current command buffer should continue.If clear, no comparison takes place.
	20	**Reserved**
	19:8	**Reserved**
	7:0	**DWord Length** Format: =n Total Length - 2

MI_CONDITIONAL_BATCH_BUFFER_END

		<table><tr><th>Value</th><th>Name</th><th>Project</th></tr><tr><td>1h</td><td>Excludes DWord (0,1) **[Default]**</td><td>CHV, BSW</td></tr></table>
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at Semaphore Address is greater than this dword, the execution of the command buffer should continue.
2	31:3	**Compare Address** <table><tr><td>Format:</td><td>GraphicsAddress[31:3]</td></tr></table> Qword address to fetch compare Mask (DW0) and Data Dword(DW1) from memory. HW will do AND operation on Mask(DW0) with Data Dword(DW1) and then compare the result against Semaphore Data Dword
	2:0	**Reserved** <table><tr><td>Format:</td><td>MBZ</td></tr></table>
3 **Project:** CHV, BSW	31:16	**Reserved** <table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table>
	15:0	**Compare Address High** <table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table> This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space

MI_CONDITIONAL_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_CONDITIONAL_BATCH_BUFFER_END command is used to conditionally terminate the execution of commands stored in a batch buffer initiated using a MI_BATCH_BUFFER_START command. Termination of second level batch buffer due to this command will also terminate the parent/first level batch buffer.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 36h MI_CONDITIONAL_BATCH_BUFFER_END Format: OpCode
	22	**Use Global GTT** Default Value: 0h If set, this command will use the global GTT to translate the **Compare Address** and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used to translate the **Compare Address**.
	21	**Compare Semaphore** Default Value: 0h If set, the value from the Compare Data Dword is compared to the value from the Compare Address in memory. If the value at Compare Address is greater than the Compare Data Dword, execution of current command buffer should continue. If clear, the parser will continue to the next command and not exit the batch buffer.
	20	**Reserved**
	19:8	**Reserved**
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 1h — Name: **[Default]** — Project: CHV, BSW
1	31:0	**Compare Data Dword** Data dword to compare memory. The Data dword is supplied by software to control execution of the command buffer. If the compare is enabled and the data at Compare Address is greater than this dword, the execution of the command buffer should continue.

MI_CONDITIONAL_BATCH_BUFFER_END

2	31:3	**Compare Address** Format: GraphicsAddress[31:3] Qword address to fetch Data Dword(DW0) from memory. HW will compare the Data Dword(DW0) with Compare Data Dword
	2:0	**Reserved** Format: MBZ
3 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW
	15:0	**Compare Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of Gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address 1h – Global Graphics Address – This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address 1h – Global Graphics Address – This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_COPY_MEM_MEM

	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3 Format: =n Total Length - 2
1	31:2	**Destination Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved**
2	31:16	**Reserved**
	15:0	**Destination Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register
3	31:2	**Source Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved**
4	31:16	**Reserved**
	15:0	**Source Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. 0h – Per Process Graphics Address 1h – Global Graphics Address: It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear.
	21	**Use Global GTT Destination** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. This bit will determine write to memory uses Global or Per Process GTT.

MI_COPY_MEM_MEM

Value	Name	Description
0h	Per Process Graphics Address	
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

20:8 Reserved

Format:	MBZ

7:0 Dword Length

Default Value:	3
Format:	=n Total Length - 2

1..2 63:2 Destination Memory Address

Project:	All
Format:	GraphicsAddress[63:2]

Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

1:0 Reserved

Project:	All
Format:	MBZ

3..4 63:2 Source Memory Address

Project:	All
Format:	GraphicsAddress[63:2]

Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

1:0 Reserved

Project:	All
Format:	MBZ

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address 1h – Global Graphics Address – This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h – Per Process Graphics Address 1h – Global Graphics Address – This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_COPY_MEM_MEM

	20:8	**Reserved**
	7:0	**DWord Length** Default Value: 3 Format: =n Total Length - 2
1	31:2	**Destination Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved**
2	31:16	**Reserved**
	15:0	**Destination Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register
3	31:2	**Source Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved**
4	31:16	**Reserved**
	15:0	**Source Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register

MI_COPY_MEM_MEM

MI_COPY_MEM_MEM

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 2

The MI_COPY_MEM_MEM command reads a DWord from memory and stores the value of that DWord to back to memory. The source and destination addresses are specified in the command. The command temporarily halts command execution.

Programming Notes

This command should not be used within a "non_privilege"batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Eh MI_MEM_TO_MEM Format: OpCode
	22	**Use Global GTT Source** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value 0h: Per Process Graphics Address Value 1h: Global Graphics Address — This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Use Global GTT Destination** It is allowed for this bit to be set when executing this command from a privileged (secure) batch buffer or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. Value 0h: Per Process Graphics Address Value 1h: Global Graphics Address — This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_COPY_MEM_MEM

	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 3 Format: =n Total Length - 2
1	31:2	**Destination Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value fetched specified in the DWord address above will be written. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved** Project: All Format: MBZ
2	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Destination Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register
3	31:2	**Source Memory Address** Project: All Format: GraphicsAddress[31:2] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[31:2] for a DWord register
	1:0	**Reserved** Project: All Format: MBZ
4	31:16	**Reserved**

MI_COPY_MEM_MEM		
	15:0	**Source Memory Address High** Project: All Format: GraphicsAddress[47:32] Surface Type: MMIO Register This field specifies the address of the memory location where the value is located that will be written to the address below. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[47:32] for a DWord register

MI_DISPLAY_FLIP

MI_DISPLAY_FLIP

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_DISPLAY_FLIP command is used to request a specific display plane to switch (flip) to display a new buffer. The buffer is specified with a starting address and pitch. The tiled attribute of the buffer start address is programmed as part of the packet.

The operation this command performs is also known as a "display flip request" operation - in that the flip operation itself will occur at some point in the future. This command specifies when the flip operation is to occur: either synchronously with vertical retrace to avoid tearing artifacts

Programming Notes

This command simply requests a display flip operation -- command execution then continues normally. There is no guarantee that the flip (even if asynchronous) will occur prior to subsequent commands being executed. (Note that completion of the MI_FLUSH_DW command does not guarantee that outstanding flip operations have completed). The MI_WAIT_FOR_EVENT command must be used to provide this synchronization to avoid back to back MI_DISPLAY_FLIP commands to the same display plane - by pausing command execution until a pending flip has actually completed. This synchronization can also be performed by use of the Display Flip Pending hardware status. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.

After a display flip operation is requested, software is responsible for initiating any required synchronization with subsequent buffer clear or blitter operations. For multi-buffering (e.g., double buffering) operations, this will typically require updating SURFACE_STATE or the binding table to change the blitter (back) buffer. In addition, prior to any subsequent clear or blitter operations, software must typically ensure that the new blitter buffer is not actively being displayed. Again, the MI_WAIT_FOR_EVENT command or Display Flip Pending hardware status can be used to provide this synchronization. See Display Flip Synchronization in the Device Programming Interface chapter of MI Functions.

The display buffer command uses the X and Y offset for the tiled buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For tiled buffers, the display subsystem uses the X and Y offset in generation of the final request to memory. The offset is always updated on the next vblank for both Synchronous and Asynch Flips. It is not necessary to have a flip enqueued to update the X and Y offset

The display buffer command uses the linear DWord offset for the linear buffers from the Display Interface registers. Software is allowed to change the offset via the MMIO interface irrespective of the flip commands enqueued in the command stream. For linear buffers, the display subsystem uses the Dword offset in generation of the final request to memory.

- For synchronous flips the offset is updated on the next vblank. It is not necessary to have a sync flip enqueued to update the DWord offset.
- Linear memory does not support asynchronous flips.

Events must be unmasked in the Display Engine Render Response Mask Register (DE RRMR 0x44050) prior to

MI_DISPLAY_FLIP

waiting for them with a MI_WAIT_FOR_EVENT command, or in the case of flips or scanlines, prior to starting the flip or loading the scanline. Unmasked events will wake command streamer as they occur, so for improved power savings it is recommended to only unmask events that are required. Programming the DE RRMR register can be done through MMIO or a LOAD_REGISTER_IMMEDIATE command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 14h MI_DISPLAY_FLIP Format: OpCode
	22	**Async Flip Indicator** Format: Enable This bit should always be set if DW2 [1:0] == '01' (async flip). This field is required due to HW limitations. This bit is used by the blitter pipe while DW2 is used by the display hardware.
	21:19	**Display (Plane) Select** This field selects which display plane is to perform the flip operation. Value: 0h — Name: Display Plane A Value: 1h — Name: Display Plane B Value: 2h — Name: Display Sprite A Value: 3h — Name: Display Sprite B Value: 4h — Name: Display Plane C Value: 5h — Name: Display Sprite C
	18:17	**Reserved**
	16	**Override Pipe/Plane** Project: CHV, BSW Value: 1h — Name: Use 15:8 bits — Description: Value: 0h — Name: Use bits 21:19 — Description: ("client mode")
	15:13	**Reserved** Format: MBZ
	12:8	**Display Plane Select** Project: CHV, BSW

MI_DISPLAY_FLIP

Value	Name
00000b	Display Plane A
00001b	Display Plane B
00010b	Display Plane C
00011b	Reserved
00100b	Display Sprite A
00101b	Display Sprite B
00110b	Display Sprite C
00111b	Display Sprite A2
01000b	Display Sprite B2
01001b	Display Sprite C2
01010b	Display Sprite A3
01011b	Display Sprite B3
01100b	Display Sprite C3
01101b	Reserved
01110b	Reserved
01111b	Reserved
1xxxxb	Remaining encodings reserved

7:0 **DWord Length**

Format:	=n Total Length - 2

For Synchronous Flips and Asynchronous Flips, this field must be programmed to 1h for a total length of 3.

Value	Name	Project	Exists If
0h	Excludes DWord (0,1) **[Default]**		
1h			([Flip Type]!='Stereo 3D Flip')
2h		CHV, BSW	([Flip Type]=='Stereo 3D Flip')

1 31 **Reserved**

Project:	CHV, BSW

30:16 **Reserved**

Project:	All
Format:	MBZ

15:6 **Reserved**

Project:	All

5:1 **Reserved**

Project:	All

MI_DISPLAY_FLIP

		Format: MBZ
	0	**Tile Parameter** Project: CHV, BSW Format: Enable For Asynchronous Flips, this parameter cannot be changed. All the flips in a flip chain should maintain the same tile parameter as programmed with the last synchronous flip or direct thru MMIO.

Value	Name	Description
0h	Linear **[Default]**	For Syncronous Flips Only
1h	Tiled X	

Programming Notes
Performing a synchronous or asynchronous flip will drop any previous synchronous flip that has not yet completed.

2	31:12	**Display Buffer Base Address** Project: All Format: GraphicsAddress[31:12] This field specifies Bits 31:12 of the Graphics Address of the new display buffer.

Programming Notes
The Display buffer must reside completely in Main Memory.
This address is always translated via the global (rather than per-process) GTT

	11:3	**Reserved** Project: All Format: MBZ
	2	**Reserved** Project: CHV, BSW
	1:0	**Flip Type** Project: CHV, BSW This field specifies whether the flip operation should be performed asynchronously to vertical retrace.

Value	Name	Description	Project
00b	Sync Flip **[Default]**	The flip will occur during the vertical blanking interval - thus avoiding any tearing artifacts.	All
01b	Async Flip	The flip will occur "as soon as possible" - and may exhibit tearing artifacts	All
1b	Reserved		All

Programming Notes

MI_DISPLAY_FLIP

<table>
<tr><td></td><td></td><td>

The Display Buffer Pitch and Tile parameter cannot be changed for asynchronous flips (i.e., the new buffer must have the same pitch/tile format as the previous buffer).
Async flips are supported on X-Tiled Frame buffers only.
For Asynch Flips the Buffers used must be 32KB aligned.
Asynch flips are supported on Display Planes A and B and C only.

</td></tr>
<tr><td rowspan="4">3
Project:
CHV, BSW</td><td>31:12</td><td>Reserved</td></tr>
<tr><td>11:3</td><td>Reserved
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>Reserved
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
</table>
</td></tr>
<tr><td>1:0</td><td>Flip Type
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
</table>
This field specifies whether the flip operation should be performed asynchronously to vertical retrace.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00b</td><td>Sync Flip
[Default]</td><td>The flip will occur during the vertical blanking interval - thus avoiding any tearing artifacts.</td><td>All</td></tr>
<tr><td>01b</td><td>Async Flip</td><td>The flip will occur "as soon as possible" - and may exhibit tearing artifacts</td><td>All</td></tr>
</table>
Programming Notes

The Display Buffer Pitch and Tile parameter cannot be changed for asynchronous flips (i.e., the new buffer must have the same pitch/tile format as the previous buffer).
Async flips are supported on X-Tiled Frame buffers only.
For Asynch Flips the Buffers used must be 32KB aligned.
Asynch flips are supported on Display Planes A and B and C only.

</td></tr>
</table>

MI_FLUSH_DW

MI_FLUSH_DW

Project:	CHV, BSW
Source:	VideoEnhancementCS
Length Bias:	2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:

- Flush any dirty data to memory.
- Invalidate the TLB cache inside the hardware

Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved** Project: All
	21	**Store Data Index** Project: All Format: U1 **Description** **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).
	20:19	**Reserved**
	18	**TLB Invalidate** Project: All Format: U1

MI_FLUSH_DW

Description
If ENABLED, all TLBs belonging to Video Enhancement Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.
If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.

17 **Reserved**

Project:	CHV, BSW
Format:	MBZ

16 **Reserved**

Project:	All
Format:	MBZ

15:14 **Post-Sync Operation**

Project:	All

Value	Name	Description	Project
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.	All
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address	All
2h	Reserved	Reserved	All
3h	Write TIMESTAMP register	Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.	

Programming Notes
If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

13:10 **Reserved**

Project:	All
Format:	MBZ

9 **Reserved**

8 **Notify Enable**

Project:	All
Format:	U1

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

MI_FLUSH_DW

7 — Reserved

Project:	All
Format:	MBZ

6 — Reserved

Project:	CHV, BSW

5:0 — DWord Length

Project:	All
Format:	=n Total Length - 2

Value	Name	Project
3h	Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]**	CHV, BSW

1 — 31:3 — Address

Project:	All
Format:	GraphicsAddress[31:3]U28

This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.

2 — Destination Address Type

Project:	All

Defines address space of Destination Address

Value	Name	Description	Project
0h	PPGTT	Use PPGTT address space for DW write	All
1h	GGTT	Use GGTT address space for DW write	All

Programming Notes
Ignored if "No write" is the selected in Operation.

1:0 — Reserved

Project:	All
Format:	MBZ

2 — Project: CHV, BSW — 31:16 — Reserved

Project:	CHV, BSW
Format:	MBZ

15:0 — Address High

Project:	CHV, BSW
Format:	GraphicsAddress[47:32]U64

This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space

MI_FLUSH_DW		
3..4 **Project:** CHV, BSW	31:0	**Immediate Data** Project: CHV, BSW This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FLUSH_DW

MI_FLUSH_DW

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to: Flush any dirty data to memory. Invalidate the TLB cache inside the hardware
Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved** Project: All Format: U1
	21	**Store Data Index** Project: CHV, BSW Format: U1 Description: **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT). Project: CHV, BSW
	20:19	**Reserved** Project: All Format: MBZ
	18	**TLB Invalidate** Project: CHV, BSW Format: U1

MI_FLUSH_DW

		Description If ENABLED, all TLBs belonging to Blitter Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h. If GFX_MODE (0x229c) bit 13, this command will cause a config write to MMIO register space with the address 0x4f100.
	17	**Reserved** Project: CHV, BSW Format: MBZ
	16	**Reserved** Project: All Format: MBZ
	15:14	**Post-Sync Operation** Project: CHV, BSW BitFieldDesc **Value** / **Name** / **Description** 0h / No Write / No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc. 1h / Write Immediate Data QWord / Write the QWord containing Immediate Data Low, High DWs to the Destination Address 2h / Reserved / Reserved 3h / Write TIMESTAMP Register / Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'. **Programming Notes** If executed in a non-secure batch buffer, the address given is in a PPGTT address space. If in a secure ring or batch, the address given is in GGTT space.
	13:10	**Reserved** Project: All Format: MBZ
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8	**Notify Enable** Project: CHV, BSW Format: U1 If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

MI_FLUSH_DW

	7:6	**Reserved**
	5:0	**DWord Length** Project: All Format: =n Total Length - 2 Value 3h: Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]** — Project: CHV, BSW
1	31:3	**Address** Project: CHV, BSW Format: GraphicsAddress[31:3]U28 This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.
	2	**Destination Address Type** Project: All Defines address space of Destination Address 0h PPGTT: Use PPGTT address space for DW write — Project: All 1h GGTT: Use GGTT address space for DW write — Project: All **Programming Notes**: Ignored if "No write" is the selected in Operation.
	1:0	**Reserved**
2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]U64 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space
3..4 **Project:** CHV, BSW	31:0	**Immediate Data** Project: CHV, BSW This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_FLUSH_DW

MI_FLUSH_DW

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MI_FLUSH_DW command is used to perform an internal "flush" operation. The parser pauses on an internal flush until all drawing engines have completed any pending operations. In addition, this command can also be used to:Flush any dirty data to memory. Invalidate the TLB cache inside the hardware Usage note: After this command is completed with a Store DWord enabled, CPU access to graphics memory will be coherent (assuming the Render Cache flush is not inhibited).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 26h MI_FLUSH_DW
	22	**Reserved** Project: CHV, BSW
	21	**Store Data Index** Project: CHV, BSW Format: U1 **Description** / **Project**: **Ring Buffer Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). **Execlist Mode Scheduling:** This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT). — CHV, BSW
	20:19	**Reserved** Format: MBZ
	18	**TLB Invalidate** Project: CHV, BSW Format: U1 If ENABLED, all TLBs belonging to Video Engine will be invalidated once the flush operation is complete. This bit is only valid when the Post-Sync Operation field is a value of 1h or 3h.

MI_FLUSH_DW

17 **Reserved**

Project:	CHV, BSW
Format:	MBZ

16 **Reserved**

Format:	MBZ

15:14 **Post-Sync Operation**

Project:	CHV, BSW

BitFieldDesc

Value	Name	Description
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.
1h	Write Immediate Data	HW implicitly detects the Data size to be Qword or Dword to be written to memory based on the command dword length programmed . When Dword Length indicates Qword, Writes the QWord containing Immediate Data Low, High DWs to the Destination Address . When Dword Length indicates Dword, Writes the DWord containing Immediate Data Low to the Destination Address
2h	Reserved	Reserved
3h		Write the TIMESTAMP register to the Destination Address. The upper 28 bits of the TIMESTAMP register are tied to '0'.

13:10 **Reserved**

Project:	All
Format:	MBZ

9 **Reserved**

Project:	CHV, BSW
Format:	MBZ

8 **Notify Enable**

Project:	CHV, BSW
Format:	U1

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

7 **Video Pipeline Cache invalidate**

Project:	CHV, BSW
Format:	U1

Enable the invalidation of the video cache at the end of this flush

6 **Reserved**

MI_FLUSH_DW

		Project: CHV, BSW
	5:0	**DWord Length** Format: =n Total Length - 2 Value 3h: Excludes DWord (0,1) = 2 for DWord, 3 for QWord **[Default]** (CHV, BSW)
1	31:3	**Address** Format: GraphicsAddress[31:3]U28 This field specifies Bits 31:3 of the Address where the DWord or QWord will be stored. Note that the address can only be QWord aligned, irrespective of data size.
	2	**Destination Address Type** Defines address space of Destination Address 0h: PPGTT — Use PPGTT address space for DW write 1h: GGTT — Use GGTT address space for DW write **Programming Notes**: Ignored if "No write" is the selected in Operation.
	1:0	**Reserved** Format: MBZ
2 **Project:** CHV, BSW	31:16	**Reserved** Project: CHV, BSW Format: MBZ
	15:0	**Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]U64 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space
3..4 **Project:** CHV, BSW	31:0	**Immediate Data** Project: CHV, BSW This field specifies the DWord value to be written to the targeted location. DW2 is the lower DW if QW is desired. Only valid when 15:14 in header is set to 1h To avoid hitting a known hardware bug, drivers cannot send a QW write when bit 5 of the address is '1'

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Project:	CHV, BSW
Source:	VideoEnhancementCS
Length Bias:	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

- The behavior of this command is controlled by Dword 3, Bit 8 (Disable Register Access) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.
- If this command is executed from a batch buffer then the behavior of this command is controlled by Dword 0, Bit 8 (Security Indicator) of the BATCH_BUFFER_START Command. If the batch buffer is non-secure then the command stream converts this command to a NOOP.
- The following addresses should NOT be used for LRIs
    1. 0x8800 - 0x88FF
    2. >= 0x40000

Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode
	22:12	**Reserved**
	11:8	**Byte Write Disables** Project: All Format: Enable[4] (bit 8 corresponds to Data DWord [7:0]). Range: Must specify a valid register write operation If [11:8] is '1111b', then the register write will not occur. If [11:8] is '0000b', then the register DW will be updated. Any other value, the behavior will be specifically specified by the register or the behavior is undefined.

MI_LOAD_REGISTER_IMM

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Project: All Format: =n Total Length - 2
1	31:23	**Reserved** Project: All Format: MBZ
	22:2	**Register Offset** Project: All Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).Mapped **Programming Notes** / **Project** Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. / CHV, BSW
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.

Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 22h MI_
	22:12	**Reserved** Project: All Format: MBZ
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range: Must specify a valid register write operation If [11:8] is '1111b', then the register write will not occur. If [11:8] is '0000b', then the register DW will be updated. Any other value, the behavior will be specifically specified by the register or the behavior is undefined.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Offset** Format: U21 Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).

MI_LOAD_REGISTER_IMM

		Programming Notes / **Project** Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. / CHV, BSW
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range).
Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

Programming Notes

A stalling flush must be sent down pipeline before issuing this command. The behavior of this command is controlled by Dword 3, Bit 8 (Disable Register Access) of the RINGBUF register. If this command is disallowed then the command stream converts it to a NOOP.
If this command is executed from a BB then the behavior of this command is controlled by Dword 0, Bit 8 (Security Indicator) of the BATCH_BUFFER_START Command. If the batch buffer is insecure then the command stream converts this command to a NOOP. Note that the corresponding ring buffer must allow a register update for this command to execute.
To ensure this command gets executed before upcoming commands in the ring, either a stalling pipeControl should be sent after this command, or MMIO 0x20C0 bit 7 should be set to 1.
When base address of 0x180000 is added to the Register Offset, when executed will result in updating of the register in the other GT in GTB mode of operation then the GT from which this instruction is executed. When this instruction is executed by Command Streamer with COREID-0 will result in updating the register in GT with COREID-1 and vice versa, when base address of 0x180000 is added to the register offset.
The following addresses should NOT be used for LRIs:

1. 0x8800 - 0x88FF
2. >= 0xC0000

Limited LRI cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each LRI.

Programming an MMIO register is equivalent to programming a non-pipeline state to the hardware and hence an explicit stalling flush needs to be programmed prior to programming this command. However for certain MMIO registers based on their functionality doing an explicit stalling flush is exempted. Listed below are the exempted registers.

- 3DPRIM_END_OFFSET - Auto Draw End Offset [CHV, BSW]
- 3DPRIM_START_VERTEX - Load Indirect Start Vertex [CHV, BSW]
- 3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count [CHV, BSW]
- 3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count [CHV, BSW]
- 3DPRIM_START_INSTANCE - Load Indirect Start Instance [CHV, BSW]
- 3DPRIM_BASE_VERTEX - Load Indirect Base Vertex [CHV, BSW]

Writes to the range 0x9400-0x97FF must be either be avoided, or serialized with a read (e.g. STORE_REGISTER_MEM) between them.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode
	22:13	**Reserved** Format: MBZ
	12	**Reserved** Project: CHV, BSW
	11:8	**Byte Write Disables** Format: Enable[4] Bit 8 corresponds to Data DWord [7:0] Range: Must specify a valid register write operation If [11:8] is '1111b', then this command will behave as a NOOP. Otherwise, the value is forwarded to the destination register.
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Offset** Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord** Mask: Bytes Write Disables Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_IMM

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MI_LOAD_REGISTER_IMM command requests a write of up to a DWord constant supplied in the command to the specified Register Offset (i.e., offset into Memory-Mapped Register Range). The register is loaded before the next command is executed.
Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 22h MI_LOAD_REGISTER_IMM Format: OpCode
	22:12	**Reserved**
	11:8	**Byte Write Disables** Format: Enable[4] (bit 8 corresponds to Data DWord [7:0]). Range: Must specify a valid register write operation If [11:8] is '1111b', then the register write will not occur. If [11:8] is '0000b', then the register DW will be updated. Any other value, the behavior will be specifically specified by the register or the behavior is undefined.
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:23	**Reserved**
	22:2	**Register Offset** Format: MmioAddress[22:2] This field specifies bits [22:2] of the offset into the Memory Mapped Register Range (i.e., this field specifies a DWord offset).Mapped Programming Notes: Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. — Project: CHV, BSW

MI_LOAD_REGISTER_IMM		
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord** Format: U32 FormatDesc This field specifies the DWord value to be written to the targeted location.

MI_LOAD_REGISTER_MEM

MI_LOAD_REGISTER_MEM

Project: CHV, BSW

Source: RenderCS, BlitterCS, VideoCS, VideoEnhancementCS

Length Bias: 2

The MI_LOAD_REGISTER_MEM command requests from a memory location and stores that DWord to a register.

Programming Notes

The command temporarily halts commands that will cause cycles down the 3D pipeline.

The following addresses should NOT be used for MMIO writes:

- 0x8800 - 0x88FF
- >= 0xC0000

Limited MMIO writes cycles to the Display Engine 0x40000-0xBFFFF) are allowed, but must be spaced to allow only one pending at a time. This can be done by issuing an SRM to the same address immediately after each MMIO write.

Any updates to the memory location exercised by this command must be ensured to be coherent in memory prior to programming of this command. This must be achieved by programming MI_ATOMIC (write to scratch space) with "CS STALL" set prior to programming of this command. Example: MI_STORE_REGISTE_MEM (0x2288, 0x2CF0_0000) .... MI_ATOMIC (MOV, Dummy data, Scratch Address) MI_LOAD_REGISTER_MEM(0x2288, 0x2CF0_0000)

This command should not be used within a non-privilege batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation.

This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.

Writes to the range 0x9400-0x97FF must be either be avoided, or serialized with a read (e.g. STORE_REGISTER_MEM) between them.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 29h MI_LOAD_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean This bit if set when executing from a non-privileged batch buffer will be treated as privilege access violation. It is allowed for this bit to be clear when executing this command from a

MI_LOAD_REGISTER_MEM

		privileged (secure) batch buffer or ring buffer. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21	**Async Mode Enable** If this bit is set then the command stream will not wait for completion of this command before executing the next command. Please refer to the LOAD_INDIRECT and Predicate registers for usage of this bit.
	20	**Reserved** Project:
	19:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: Excludes DWord (0,1) **[Default]** — Project: CHV, BSW
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Address** Format: MMIOAddress[22:2] This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. Programming Notes: Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. — Project: CHV, BSW — Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
	1:0	**Reserved** Format: MBZ
2..3 **Project:** CHV, BSW	63:2	**Memory Address** Project: CHV, BSW Format: GraphicsAddress[63:2] This field specifies the address of the memory location where the register value specified in the DWord above will read from. The address specifies the DWord location of the data. Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved**

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_REG

Project:	CHV, BSW
Source:	CommandStreamer
Length Bias:	2

The MI_LOAD_REGISTER_REG command reads from a source register location and writes that value to a destination register location.
Any offset that is to a destination outside of the GT core will allow the parser to continue once the cycle is at the GT boundry and not destination. Any other address will ensure the destination is updated prior to parsing the next command

Programming Notes

- The command temporarily halts commands that will cause cycles down the 3D pipeline.
- Destination register with mask implemented will not get updated unless the value read from source register has the bits corresponding to the mask bits set. Note that any mask implemented register when read returns "0" for the bits corresponding to mask location. When the source and destination are mask implemented registers, destination register will not get updated with the source register contents.
- This command is not allowed to update the privilege register range when executed from a non-privilege batch buffer.
- Writes to the range 0x9400-0x97FF must be either be avoided, or serialized with a read (e.g. STORE_REGISTER_MEM) between them.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Ah Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 1h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:23	**Reserved**
	22:2	**Source Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ.

MI_LOAD_REGISTER_REG

		Programming Notes / **Project** / **Source** Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. / CHV, BSW / BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
	1:0	**Reserved** Format: MBZ
2	31:23	**Reserved** Format: MBZ
	22:2	**Destination Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be written to. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. **Programming Notes** / **Project** / **Source** Bits 22:18 must be zero. Setting these bits could cause a hang due to PM requesting a stop at the same time the request is going to a MMIO space outside the GT core. / CHV, BSW / BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
	1:0	**Reserved** Format: MBZ

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_EXCL

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is *outside* this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe.

Note: The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 13h MI_LOAD_SCAN_LINES_EXCL Format: OpCode
	22	**Reserved** Project: All Format: MBZ
	21:19	**Display Pipe Select** Project: All Format: U3 This field selects which Display Engine (pipe) this command is targeting. Value / Name / Project 0h / Display Pipe A / All 1h / Display Pipe B / All 4h / Display Pipe C / All
	18:17	**Reserved**
	16:6	**Reserved**
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2

MI_LOAD_SCAN_LINES_EXCL		
1	31:16	**Start Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]
	15:0	**End Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_EXCL

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

The MI_LOAD_SCAN_LINES_EXCL command is used to initialize the Scan Line Window registers for a specific Display Pipe. If the display refresh is **outside** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while outside). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display pipe. **Note:** The two scan-line numbers are inclusive. If programmed to the same values, that single line defines the region in question. Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 13h MI_LOAD_SCAN_LINES_EXCL Format: OpCode
	22	**Reserved**
	21:19	**Display (Plane) Select** Format: U3 This field selects which display plane is to perform the scanline operation. Value / Name: 0h Display Plane A 1h Display Plane B 2h Reserved 3h Reserved 4h Display Plane C 5h Reserved
	18:17	**Reserved**
	16:6	**Reserved**
	5:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).

MI_LOAD_SCAN_LINES_EXCL		
1	31:29	**Reserved** Format: MBZ
	28:16	**Start Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line Window.
	15:13	**Reserved** Format: MBZ
	12:0	**End Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is *within* this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while inside of the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.

Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 12h MI_LOAD_SCAN_LINES_INCL Format: OpCode
	22	**Reserved** Project: All Format: MBZ
	21:19	**Display Pipe Select** Project: All Format: U3 This field selects which Display Engine (pipe) this command is targeting. Value / Name: 0h Display Pipe A 1h Display Pipe B 4h Display Pipe C
	18:17	**Reserved**
	16:6	**Reserved** Project: CHV, BSW Format: MBZ
	5:0	**DWord Length** Default Value: 0h Excludes DWord (0,1)

MI_LOAD_SCAN_LINES_INCL

		Format: =n Total Length - 2
1	31:16	**Start Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the starting scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]
	15:0	**End Scan Line Number** Format: U16 In scan lines, where scan line 0 is the first line of the display frame. This field specifies the ending scan line number of the Scan Line Window. Range: [0, Display Buffer height in lines-1]

MI_LOAD_SCAN_LINES_INCL

MI_LOAD_SCAN_LINES_INCL

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_LOAD_SCAN_LINES_INCL command is used to initialize the Scan Line Window registers for a specific Display Engine. If the display refresh is **within** this window the Display Engine asserts a signal that is used by the command parser to process the WAIT_FOR_EVENT command (i.e., the parser will wait while inside the window). This command overrides the Scan Line Window defined by any previous MI_LOAD_SCAN_LINES_INCL or MI_LOAD_SCAN_LINES_EXCL commands targeting the specific display.
Always place an even number of MI_LOAD_SCAN_LINES_EXCL/INCL at a time into the ring buffer. If only a single MI_LOAD_SCAN_LINES_EXCL/INCL is desired, just add a second identical MI_LOAD_SCAN_LINES_EXCL/INCL command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 12h MI_LOAD_SCAN_LINES_INCL Format: OpCode
	22	**Reserved** Format: MBZ
	21:19	**Display (Plane) Select** Project: CHV, BSW Format: U3 This field selects which display plane is to perform the scanline operation. Value / Name: 0h Display Plane A 1h Display Plane B 2h Reserved 3h Reserved 4h Display Plane C 5h Reserved
	18:17	**Reserved**
	16:6	**Reserved** Project: CHV, BSW Format: MBZ

MI_LOAD_SCAN_LINES_INCL

	5:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31	**Reserved** Project: CHV, BSW Format: Must Be One
	30	**Rserved** Project: CHV, BSW Format: MBZ
	29	**Reserved** Format: MBZ
	28:16	**Start Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the starting scan line number of the Scan Line window.
	15:13	**Reserved** Format: MBZ
	12:0	**End Scan Line Number** Format: U13 In scan lines, where scan line 0 is the first line of the display frame. Range: [0, Display Buffer height in lines-1] This field specifies the ending scan line number of the Scan Line Window.

MI_LOAD_URB_MEM

MI_LOAD_URB_MEM

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_LOAD_URB_MEM command requests from a memory location and stores that DWord to the URB.

Programming Notes

The command temporarily halts commands that will cause cycles down the 3D pipeline.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Ch MI_LOAD_URB_MEM Format: OpCode
	22:8	**Reserved**
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: **[Default]** — Project: CHV, BSW
1	31:15	**Reserved** Format: MBZ
	14:2	**URB Address** This field specifies Bits 14:2 of the URB offset the DWord will be written in the URB. This command only supports writing below 32KB of the URB space.
	1:0	**Reserved** Format: MBZ
2..3 **Project:** CHV, BSW	63:6	**Memory Address** Project: CHV, BSW Format: GraphicsAddress[63:6] This field specifies the address of the location of where the value will be read from memory. The value must be in the first DW location of the cache line. Range = GraphicsVirtualAddress[47:6] GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	5:0	**Reserved**

MI_MATH

MI_MATH

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer.
This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.
When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Project: CHV, BSW
Source: VideoCS
Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer.
This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.
When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 2

The MI_MATH command allows software to send instructions to the ALU in the Command Streamer. This command is the means by which the ALU is accessed. ALU instructions form the data payload of the MI_MATH command. An ALU instruction takes one DWord in size. The MI_MATH DWord Length is programmed based on the number of ALU instructions included, limited only by the max DWord Length supported.

When the command streamer parses an MI_MATH command, it sends the included ALU instructions to the ALU. The ALU processes any instruction in a single clock. See the ALU section for more details.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_MATH

MI_MATH

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_MATH command allows SW to send instruction to ALU in Render Command Streamer.
MI_MATH command is the means by which ALU can be accessed. ALU instructions form the data payload of MI_MATH command, ALU instruction is dword in size. MI_MATH Dword Length should be programmed based on the number of ALU instruction packed, max number is limited by the max Dword Length supported. When MI_MATH command is parsed by command streamer it outputs the payload dwords (ALU instructions) to the ALU. ALU takes single clock to process any given instruction. Refer to B-spec "Command Streamer (CS) ALU Programming" section in Command Streamer Programming.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ah MI_MATH Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:6	**Reserved** Project: CHV, BSW Format: MBZ
	5:0	**DWord Length** Default Value: 0h Project: CHV, BSW Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**ALU INSTRUCTION 1** Format: Table Entry
2	31:0	**ALU INSTRUCTION 2** Format: Table Entry
3..n	31:0	**ALU INSTRUCTION n** Format: Table Entry

MI_NOOP

MI_NOOP

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value / Name / Description 1 / / Write th NOP_ID Register 0 / / Do not write the NOP_ID register
	21:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP

Project: CHV, BSW
Source: BlitterCS
Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP
	22	**Identification Number Register Write Enable** Project: All Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value / Name / Description / Project 0h / Disable / Do not write the NOP_ID register. / All 1h / Enable / Write the NOP_ID register. / All
	21:0	**Identification Number** Project: All Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP

Project: CHV, BSW

Source: RenderCS

Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

Performance	Project
The MI_NOOP process time is reduced to 1 clock. An example use of the improved NOOP throughput is for some multi-pass media applications where some unwanted media object commands are replaced by MI_NOOP commands without repacking the commands in a batch buffer.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0h MI_NOOP
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified, making this command an effective "no operation" function. Value / Name / Description: 0h / Disable / Do not write the NOP_ID register. 1h / Enable / Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_NOOP

MI_NOOP

Project: CHV, BSW
Source: VideoCS
Length Bias: 1

The MI_NOOP command basically performs a "no operation" in the command stream and is typically used to pad the command stream (e.g., in order to pad out a batch buffer to a QWord boundary). However, there is one minor (optional) function this command can perform - a 22-bit value can be loaded into the MI NOPID register. This provides a general-purpose command stream tagging ("breadcrumb") mechanism (e.g., to provide sequencing information for a subsequent breakpoint interrupt).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 00h MI_NOOP Format: OpCode
	22	**Identification Number Register Write Enable** Format: Enable This field enables the value in the Identification Number field to be written into the MI NOPID register. If disabled, that register is unmodified - making this command an effective "no operation" function. Value: 1 — Name: Write the NOP_ID register.
	21:0	**Identification Number** Format: U22 This field contains a 22-bit number which can be written to the MI NOPID register.

MI_PREDICATE

MI_PREDICATE

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Ch MI_PREDICATE Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:6	**Load Operation** This field controls if/how the Predicate state bit is modified. Value / Name / Description 0h / KEEP / The Predicate state bit is unmodified. 1h / Reserved / 2h / LOAD / The Predicate state bit is loaded with the combine operation result. 3h / LOADINV / The Predicate state bit is loaded with the inverted combine operation result.
	5	**Reserved** Format: MBZ
	4:3	**Combine Operation** This field controls if/how the result of the compare operation is combined with the current Predicate state bit. Value / Name / Description 0h / SET / The combine operation output the compare result unmodified. 1h / AND / The combine operation outputs the AND of the compare result and the current Predicate state bit. 2h / OR / The combine operation outputs the OR of the compare result and the current Predicate state bit. 3h / XOR / The combine operation outputs the XOR of the compare result and the current Predicate state bit.
	2	**Reserved** Format: MBZ

MI_PREDICATE

	1:0	**Compare Operation** This field controls how Data DWord 0 and Data DWord 1 fields are used to generate a compare operation result and possibly modify the PredicateData register.

Value	Name	Description
0h	TRUE	The compare operation outputs TRUE. The PredicateData register is unmodified.
1h	FALSE	The compare operation outputs FALSE. The PredicateData register is unmodified.
2h	SRCS_EQUAL	(MItemp0 - MItemp1) is computed and loaded into the PredicateData register. The compare operation outputs (MItemp0 == MItemp1).
3h	DELTAS_EQUAL	(MItemp0 - MItemp1) is computed and compared to the PredicateData register. If the values are equal, the compare operation outputs TRUE, otherwise it outputs FALSE. The PredicateData register is unmodified.

MI_REPORT_HEAD

MI_REPORT_HEAD

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.

When the **Per-Process Virtual Address Space and Execlist Enable bit** is reset: The location written is relative to the address programmed in the Hardware Status Page Address Register. When the **Execlist Enable** is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Project: CHV, BSW
Source: BlitterCS
Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location.
When the **Execlist Enable** bit is reset:
The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).
When the **Execlist Disable** is clear, the head pointer will be reported to the PP HW Status Page.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD
	22:0	**Reserved** Project: All Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Project: CHV, BSW

Source: RenderCS

Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the active ring buffer to be written to a cacheable (snooped) system memory location. When Execlist Enable is set, the head pointer will be reported to the PP HW Status Page. The location written is relative to the address programmed in the Hardware Status Page Address Register.

Programming Notes

This command must not be executed from a Batch Buffer. (Refer to the description of the HWS_PGA register.)

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_REPORT_HEAD

MI_REPORT_HEAD

Project: CHV, BSW
Source: VideoCS
Length Bias: 1

The MI_REPORT_HEAD command causes the Head Pointer value of the ring buffer to be written to a cacheable (snooped) system memory location.When the Per-Process Virtual Address Space and Execlist Enable bitis reset:The location written is relative to the address programmed in the Hardware Status Page Address Register. When the Execlist Enable is set, the head pointer will be reported to the PP HW Status Page.

Programming Notes

This command must not be executed from a Batch Buffer (Refer to the description of the HWS_PGA register).

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 07h MI_REPORT_HEAD Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_RS_CONTROL

MI_RS_CONTROL

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	1

The MI_RS_CONTROL command is used to start or stop the Resource Streamer.

Programming Notes

- This command must be programmed only inside a Resource Streamer enabled batch buffer.
- This command provides means to selectively disable or enable Resource Streamer for set of commands in a Resource Streamer enabled batch buffer
- On re-enabling the Resource Streamer through this command, command streamer will start Resource Streamer on the next non-sync command of the batch buffer.
- This command status is render context save/restored during context switching.
- The scope of MI_RS_CONTROL is within the batch buffer it is programmed, it doesn't get carried to the following chained batch buffer or second level batch buffer. RS control status goes back to default mode of Resource Streamer Enabled on all batch buffer arbitration boundaries. Batch buffer arbitration boundaries includes calling a chained or a second level batch buffer through MI_BATCH_BUFFER_START command or terminating a batch buffer through MI_BATCH_BUFFER_END command.
- Example:

1. MI_BATCH_START (Primary batch buffer with RS enable)
2. Command 1 --> CS starts RS
3. Command 2
4. :
5. MI_RS_CONTROL (stop option) -> RS will stop on this command, CS sets RS control status to STOP.
6. Command 3
7. MI_BATCH_START (2nd level batch with RS enable not set, RS control status gets reset to default status of START)
8. :
9. MI_BATCH_END (Second Level Batch End)
10. Command 4 --> CS starts RS here as RS control flag gets reset to START at step-7
11. MI_BATCH_BUFFER_END

Workaround

Workaround:
Due to known HW issue "Resource Streamer Control" status of MI_RS_CONTROL command is not context save/restored across context switches. SW must ensure all pool allocations (3DSTATE_BINDING_TABLE_POOL_ALLOC, 3DSTATE_GATHER_POOL_ALLOC, 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC) are disabled and no Resource Streamer specific commands are programmed when the "Resource Streamer Control" is programmed to "Stop".

MI_RS_CONTROL

Example:
……
MI_RS_CONTROL (Stop Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Disable)
3DSTATE_GATHER_POOL_ALLOC (Gather Pool Disable)
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC (Constant Buffer Pool Disable)
//Following Commands must not be programmed
//3DSTATE_BINDING_TABLE_EDIT_*
//3DSTATE_GATHER_CONSTANT_*
//3DSTATE_DX9_CONSTANTF_*
……
MI_RS_CONTROL (Start Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enable)
3DSTATE_GATHER_POOL_ALLOC (Gather Pool Enable)
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC (Constant Buffer Pool Enable)
……

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:29</td><td>Command Type
<table><tr><td>Default Value:</td><td>0h MI_COMMAND</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>28:23</td><td>MI Command Opcode
<table><tr><td>Default Value:</td><td>06h MI_RS_CONTROL</td></tr><tr><td>Format:</td><td>OpCode</td></tr></table></td></tr>
<tr><td></td><td>22:1</td><td>Reserved
<table><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
<tr><td></td><td>0</td><td>Resource Streamer Control
<table><tr><td>Format:</td><td>U1</td></tr></table>This bit specifies whether the command is starting or stopping the Resource Streamer.
<table><tr><th>Value</th><th>Name</th><th>Description</th></tr><tr><td>0h</td><td>Stop</td><td>Stop and disable the Resource Streamer</td></tr><tr><td>1h</td><td>Start</td><td>Start and enable the Resource Streamer</td></tr></table></td></tr>
</table>

MI_RS_STORE_DATA_IMM

MI_RS_STORE_DATA_IMM

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_RS_STORE_DATA_IMM command requests a write of the DWord constant supplied in the packet to the specified Memory Address.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Bh Format: OpCode MI_RS_STORE_DATA_IMM
	22	**Reserved** Project: CHV, BSW Format: MBZ
	21	**Reserved**
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Format: =n Total Length - 2. Excludes DWord (0,1).
1..2 **Project:** CHV, BSW	63:2	**Destination Address** Project: CHV, BSW Format: GraphicsAddress[63:2] This field specifies Bits 47:2 of the Address where the DWord will be stored. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. When render engine is PPGTT enabled this Address is translated using PPGTT, else GGTT is used for translation.
	1	**Reserved** Project: CHV, BSW Format: MBZ

MI_RS_STORE_DATA_IMM		
	0	**Core Mode Enable** Project: CHV, BSW If this bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data.
3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_SIGNAL

Project: CHV, BSW

Source: CommandStreamer

Length Bias: 2

This command is used to signal the target engine stating the memory semaphore update occurrence to one of its contexts with **Target Context ID.** MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command. MI_ATOMIC (non-posted) command will be programmed prior to this command to update the semaphore data in memory.

Programming Notes	Source
[All Command Streamers]: When SW intends to use semaphore signaling between Command streamers, it must avoid lite restores in HW by programming "Force Restore" bit to '1' in context descriptor during context submission, this is required to avoid known HW issue.	

Workaround	Project
Workaround: Post-Sync operation bit must not be set when Target Engine Select is set to RCS.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Bh MI_SEMAPHORE_SIGNAL Format: OpCode
	22	**Reserved** Format: MBZ
	21	**Post-Sync Operation** Project: CHV, BSW Source: RenderCS Value / Name / Description: 0h – No Post Sync Operation – Command is executed as usual. 1h – Post Sync Operation – MI_SEMAPHORE_SIGNAL command is executed as a pipelined PIPE_CONTROL flush command with Semaphore Signal as post sync operation. Flush completion only guarantees the workload prior to this command is pushed till Windower unit and completion of any outstanding

MI_SEMAPHORE_SIGNAL

		flushes issued prior to this command.

Programming Notes
Any desired pipeline flush operation can be achieved by programming PIPE_CONTROL command prior to this command. When this bit is set Command Streamer sends a flush down the pipe and the atomic operation is saved as post sync operation. Command streamer goes on executing the following commands. Atomic operation saved as post sync operation is executed at some point later on completion of corresponding flush issued. When this bit is set atomic semaphore signal operation will be out of order with rest of the MI commands programmed in the ring buffer or batch buffer, it will be in order with respect to the post sync operations resulting due to PIPE_CONTROL command.
This bit must not be set due to known HW issues.

Workaround
Workaround: "Post Sync Operation" bit must not be set when MI_SEMAPHPORE_SIGNAL command is programmed by GPGPU and MEDIA workloads (i.e when PIPELINE_SELECT command is set to GPGPU or MEDIA). This is to WA FFDOP CG issue.
Workaround: Post-Sync operation bit must not be set when Target Engine Select is set to RCS.

21 **Reserved**

Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

20:18 **Reserved**

Format:	MBZ

17:15 **Target Engine Select**

Project:	CHV, BSW

This field selects the target engine to which SIGNAL will be send to.

Value	Name
0h	RCS
1h	VCS0
2h	BCS
3h	VECS
4h	VCS1
5h-7h	Reserved

14:8 **Reserved**

Format:	MBZ

7:0 **DWord Length**

MI_SEMAPHORE_SIGNAL

		Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:0	**Target Context ID** **Description** In execlist based scheduling this field contains the Context ID corresponding to the context of the target engine that this command is signaling. Target engine waiting on MI_SEMPHORE_WAIT in signal mode will re-fetch the data from memory or comparison if its context ID is same as this signaled Context ID. When execlists are enabled, Target engine on receiving this Context ID sends message to the SHIM if it doesn't have the context with the same Context ID running. Message send to SHIM carries the Context ID which will be looked at by UC for rescheduling the signaled Context ID. Target engine waiting on MI_SEMAPHORE_WAIT in signal mode will fetch data from memory for comparison on receiving signal irrespective of the context id received. In ring buffer mode of scheduling this field doesn't have any relevance.

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Project:	CHV, BSW
Source:	CommandStreamer
Length Bias:	2

Description

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT. Running on the same engine is only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait. Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in CTXT_SR_CTL register.
- In ring buffer mode of scheduling or Execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- Exec-List Scheduling: CS generates semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Programming Notes	Source
Render CS Only: SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set prior to programming MI_SEMAPHORE_WAIT command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies for certain programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.	RenderCS
Render CS Only: Ring Buffer Scheduling: CS doesn't generate semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful.	RenderCS
[Ring Buffer Mode Of scheduling] [BlitterCS, VideoCS, VideoEnhancementCS, VideoCS2: Command Streamers Only]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information following MI_SEMAPHORE_WAIT command. This will ensure Page Directory information gets reprogrammed after exiting IDLE flow triggered on MI_SEMAPHORE_WAIT command. Alternatively SW can disable IDLE flows on MI_SEMAPHORE_WAIT by setting “Semaphore Wait Event IDLE Message Disable” bit in “BCS_ECOSKPD” register.	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS

MI_SEMAPHORE_WAIT

Workaround
Workaround: [All Command Streamers][Ring Buffer Mode of Scheduling]: MI_SEMAPHORE_WAIT command must be always programmed with “Wait Mode” set to “Polling Mode”

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be 1 if the **Per Process GTT Enable** bit is clear. Value 0h: Per Process Graphics Address Value 1h: Global Graphics Address — This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.
	21:18	**Reserved** Format: MBZ
	17	**Reserved** Format: MBZ
	16	**Reserved** Project: CHV, BSW
	15	**Wait Mode** This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out. Value 1h: Polling Mode — In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register. Value 0h: Signal Mode — In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.
	14:12	**Compare Operation** This field specifies the operation that will be executed to create the result that will either

MI_SEMAPHORE_WAIT

		allow the context to continue or wait. SAD = Semaphore Address Data SDD = Semaphore Data Dword

Value	Name	Description
0h	SAD_GREATER_THAN_SDD	If Indirect fetched data is greater than inline data then continue.
1h	SAD_GREATER_THAN_OR_EQUAL_SDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	SAD_LESS_THAN_SDD	If Indirect fetched data is less than inline data then continue.
3h	SAD_LESS_THAN_OR_EQUAL_SDD	If Indirect fetched data is less than or equal to inline data then continue.
4h	SAD_EQUAL_SDD	If Indirect fetched data is equalto inline data then continue.
5h	SAD_NOT_EQUAL_SDD	If Indirect fetched data is not equal to inline data then continue.
6h	Reserved	
7h	Reserved	

	11:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Format: =n Total Length - 2. Excludes DWord (0,1)
1	31:0	**Semaphore Data Dword** Format: U32 This Data dword is supplied by software to control execution of the command buffer. This value is used as part of the comparison to result in waiting or continuing in the command parser if enabled.
2..3 **Project:** CHV, BSW	63:2	**Semaphore Address** Project: CHV, BSW Format: GraphicsAddress[63:2] This field is the Graphics Memory Address of the 32-bit value for the semaphore. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form.
	1:0	**Reserved** Project: CHV, BSW

MI_SEMAPHORE_WAIT			
		Format:	MBZ

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

Description

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT, same engine only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait.
Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in BCS_CTXT_SR_CTL register.
- In ring buffer mode of scheduling or Execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- BCS always generates an interrupt to the scheduler on encountering semaphore failure.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Programming Notes

[Ring Buffer Mode Of scheduling][Video CS, Video Enhancement CS, Blitter CS]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information following MI_SEMAPHORE_WAIT command. This will ensure Page Directory information gets reprogrammed after exiting IDLE flow triggered on MI_SEMAPHORE_WAIT command. Alternatively SW can disable IDLE flows on MI_SEMAPHORE_WAIT by setting “Semaphore Wait Event IDLE Message Disable” bit in “BCS_ECOSKPD” register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.

MI_SEMAPHORE_WAIT

Value	Name	Description
0h	Per Process Graphics Address	
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

21:16 **Reserved**

15 **Wait Mode**

This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out.

Value	Name	Description
1h	Polling Mode	In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register.
0h	Signal Mode	In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.

14:12 **Compare Operation**

This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait. If the below operation is TRUE then
SAD = Semaphore Address DataSDD = Semaphore Data Dword

Value	Name	Description
0h	SAD > SDD	If Indirect fetched data is greater than inline data then continue
1h	SAD >= SDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	SAD < SDD	If Indirect fetched data is less than inline data then continue.
3h	SAD <= SDD	If Indirect fetched data is less than or equal to inline data then continue.
4h	SAD == SDD	If Indirect fetched data is equalto inline data then continue.
5h	SAD != SDD	If Indirect fetched data is not equal to inline data then continue.
6h	Reserved	
7h	Reserved	

11:8 **Reserved**

Format:	MBZ

7:0 **DWord Length**

Default Value:	2h Excludes DWord (0,1)

MI_SEMAPHORE_WAIT

		Format: =n Total Length - 2
1	31:0	**Semaphore Data Dword** Format: U32 Data dword to compare. The Data dword is supplied by software to control execution of the command buffer. If the data at Semaphore Address is greater than this dword, the execution of the command buffer continues.
2..3	63:48	**Reserved** Format: MBZ
	47:2	**Semaphore Address** Format: GraphicsAddress[47:2] This field is the Graphics Memory Address of the 32 bit value for the semaphore.
	1:0	**Reserved** Format: MBZ

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT. Running on the same engine is only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait. Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in CTXT_SR_CTL register
- . In ring buffer mode of scheduling or Execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- Exec-List Scheduling: CS generates semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Programming Notes

Render CS Only: SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set prior to programming MI_SEMAPHORE_WAIT command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies for certain programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.

Render CS Only: Ring Buffer Scheduling: CS doesn't generate semaphore wait interrupt to the scheduler when MI_SEMAPHORE_WAIT command is un-successful.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch

MI_SEMAPHORE_WAIT

buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be 1 if the **Per Process GTT Enable** bit is clear.

Value	Name	Description
0h	Per Process Graphics Address	
1h	Global Graphics Address	This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

21:16 **Reserved**

15 **Wait Mode**

This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out.

Value	Name	Description
1h	Polling Mode	In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register.
0h	Signal Mode	In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.

14:12 **Compare Operation**

This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait.

SAD = Semaphore Address Data
SDD = Semaphore Data Dword

Value	Name	Description
0h	SAD_GREATER_THAN_SDD	If Indirect fetched data is greater than inline data then continue.
1h	SAD_GREATER_THAN_OR_EQUAL_SDD	If Indirect fetched data is greater than or equal to inline data then continue.
2h	SAD_LESS_THAN_SDD	If Indirect fetched data is less than inline data then continue.
3h	SAD_LESS_THAN_OR_EQUAL_SDD	If Indirect fetched data is less than or equal to inline data then continue.
4h	SAD_EQUAL_SDD	If Indirect fetched data is equalto inline data then continue.
5h	SAD_NOT_EQUAL_SDD	If Indirect fetched data is not equal to inline data then continue.
6h	Reserved	

MI_SEMAPHORE_WAIT

		7h Reserved
	11:8	**Reserved**
	7:0	**DWord Length** Default Value: 2h Format: =n Total Length - 2. Excludes DWord (0,1)
1	31:0	**Semaphore Data Dword** Format: U32 Data dword to compare. The Data dword is supplied by software to control execution of the command buffer. If the data at **Semaphore Address** is greater than this dword, the execution of the command buffer continues.
2	31:2	**Semaphore Address** Format: GraphicsAddress[31:2] This field is the Graphics Memory Address of the 32-bit value for the semaphore.
	1:0	**Reserved**
3	31:16	**Reserved**
	15:0	**Semaphore Address High** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Project:	CHV, BSW
Source:	VideoCS
Length Bias:	2

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT, same engine only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait.
Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in VCS_CTXT_SR_CTL register.
- In ring buffer mode of scheduling or Execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- VCS always generates an interrupt to the scheduler on encountering semaphore failure.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Programming Notes

[Ring Buffer Mode Of scheduling][Video CS]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information following MI_SEMAPHORE_WAIT command. This will ensure Page Directory information gets reprogrammed after exiting IDLE flow triggered on MI_SEMAPHORE_WAIT command. Alternatively SW can disable IDLE flows on MI_SEMAPHORE_WAIT command by setting “Semaphore Wait Event IDLE Message Disable” bit in “VCS_ECOSKPD” register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. 0h: Per Process Graphics Address

MI_SEMAPHORE_WAIT

	21:16	**Reserved** Format: MBZ
	15	**Wait Mode** This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out. Value / Name / Description: 1h / Polling Mode / In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register. 0h / Signal Mode / In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.
	14:12	**Compare Operation** This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait. If the below operation is TRUE then Value / Name / Description: 0h / SAD > SDD / If Indirect fetched data is greater than inline data then continue. 1h / SAD >= SDD / If Indirect fetched data is greater than or equal to inline data then continue. 2h / SAD < SDD / If Indirect fetched data is less than inline data then continue. 3h / SAD <= SDD / If Indirect fetched data is less than or equal to inline data then continue. 4h / SAD == SDD / If Indirect fetched data is equalto inline data then continue. 5h / SAD != SDD / If Indirect fetched data is not equal to inline data then continue. 6h / Reserved / 7h / Reserved / **Programming Notes** SAD = Semaphore Address DataSDD = Semaphore Data Dword
	11:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Semaphore Data Dword**

MI_SEMAPHORE_WAIT		
		Format: U32 Data dword to compare. The Data dword is supplied by software to control execution of the command buffer. If the data at **Semaphore Address** is greater than this dword, the execution of the command buffer continues.
2	31:2	**Semaphore Address** Format: GraphicsVirtualAddress[31:2] This field is the Graphics Memory Address of the 32 bit value for the semaphore.
	1:0	**Reserved** Format: MBZ
3	31:16	**Reserved** Format: MBZ
	15:0	**Semaphore Address High** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_SEMAPHORE_WAIT

MI_SEMAPHORE_WAIT

Project:	CHV, BSW
Source:	VideoEnhancementCS
Length Bias:	2

This command supports memory based Semaphore WAIT. Memory based semaphores will be used for synchronization between the Producer and the Consumer contexts. Producer and Consumer Contexts could be running on different engines or on the same engine inside GT, same engine only possible when execlists are enabled. Producer Context implements a Signal and Consumer context implements a Wait.
Command Streamer on parsing this command fetches data from the Semaphore Address mentioned in this command and compares it with the inline Semaphore Data Dword.

- If comparison passes, the command streamer moves to the next command.
- When execlists are enabled, if comparison fails Command streamer switches out the context. Context switch can be inhibited by setting "Inhibit Synchronous Context Switch" in VECS_CTXT_SR_CTL register.
- In ring buffer mode of scheduling or execlist with "Inhibit Synchronous context Switch", if comparison fails, Command Streamer evaluates the Compare Operation based on the Wait Mode until the compare operation is true or Wait is canceled by SW.
- VECS always generates an interrupt to the scheduler on encountering semaphore failure.

MI_SEMPHORE_SIGNAL and MI_SEMAPHORE_WAIT together replace the MI_SEMAPHORE_MBOX command.

Programming Notes

[Ring Buffer Mode Of scheduling][Video Enhancement CS]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information following MI_SEMAPHORE_WAIT command. This will ensure Page Directory information gets reprogrammed after exiting IDLE flow triggered on MI_SEMAPHORE_WAIT command. Alternatively SW can disable IDLE flows on MI_SEMAPHORE_WAIT by setting “Semaphore Wait Event IDLE Message Disable” bit in “VECS_ECOSKPD” register.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 1Ch MI_SEMAPHORE_WAIT Format: OpCode
	22	**Memory Type** This bit will be ignored and treated as if clear when executing from a non-privileged batch buffer. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit *must* be '1' if the **Per Process GTT Enable** bit is clear. Value: 0h — Name: Per Process Graphics Address

MI_SEMAPHORE_WAIT

	21:16	**Reserved** Format: MBZ
	15	**Wait Mode** This bit specifies the WAIT behavior when the semaphore comparison fails and before the context is switched out. Value 1h (Polling Mode): In this mode HW periodically reads the semaphore data from memory for comparison until it is context switched out. Periodicity will be mentioned in a SEMA_WAIT_POLL register. Value 0h (Signal Mode): In this mode HW will reacquire the semaphore data from memory on receiving SIGNAL with the same Context ID. In ring buffer mode of scheduling Context ID associated with SIGNAL is ignored and always treated as a match.
	14:12	**Compare Operation** This field specifies the operation that will be executed to create the result that will either allow the context to continue or wait. If the below operation is TRUE then Value 0h (SAD > SDD): If Indirect fetched data is greater than inline data then continue. Value 1h (SAD >= SDD): If Indirect fetched data is greater than or equal to inline data then continue. Value 2h (SAD < SDD): If Indirect fetched data is less than inline data then continue. Value 3h (SAD <= SDD): If Indirect fetched data is less than or equal to inline data then continue. Value 4h (SAD == SDD): If Indirect fetched data is equalto inline data then continue. Value 5h (SAD != SDD): If Indirect fetched data is not equal to inline data then continue. Value 6h (Reserved) Value 7h (Reserved) **Programming Notes** SAD = Semaphore Address DataSDD = Semaphore Data Dword
	11:8	**Reserved**
	7:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:0	**Semaphore Data Dword**

MI_SEMAPHORE_WAIT

		Format: U32 Data dword to compare. The Data dword is supplied by software to control execution of the command buffer. If the data at **Semaphore Address** is greater than this dword, the execution of the command buffer continues.
2	31:2	**Semaphore Address** Format: GraphicsVirtualAddress[31:2] This field is the Graphics Memory Address of the 32 bit value for the semaphore.
	1:0	**Reserved**
3	31:16	**Reserved** Format: MBZ
	15:0	**Semaphore Address High** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.

MI_SET_CONTEXT

MI_SET_CONTEXT

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	2

The MI_SET_CONTEXT command is used to specify the logical context associated with the hardware context. A logical context is an area in memory used to store hardware context information, and the context is referenced via a 2KB-aligned pointer. If the (new) logical context is different (i.e., at a different memory address), the device saves the current HW context values to the current logical context address, and then restores (loads) the new logical context by reading the context from the new address and loading it into the hardware context state. If the logical context address specified in this command matches the current logical context address, this command is effectively treated as a NOOP. Specific to the Render command stream only. This command also includes some controls over the context save/restore process. The Force Restore bit can be used to refresh the on-chip device state from the same memory address if the indirect state buffers have been modified. The Restore Inhibit bit can be used to prevent the new context from being loaded at all. This must be used to prevent an uninitialized context from being loaded. Once software has initialized a context (by setting all state variables to initial values via commands), the context can then be stored and restored normally. When switching from a generic media context to a 3D context, the generic media state must be cleared via the Generic Media State Clear bit 16 in PIPE_CONTROL (or bit 4 in MI_FLUSH) before saving 3D context. MI_SET_CONTEXT commands are permitted only within a ring buffer (not within a batch buffer).

Programming Notes

- This command is legal only if Execlist Enable in the GFX_MODE register is reset. Otherwise, execlists must be used to switch context in lieu of MI_SET_CONTEXT.
- This command needs to be always followed by a single MI_NOOP instruction to workaround a silicon issue.
- [CHV, BSW]: MI_ARB_ON_OFF with 'Arbitration Enable Reset' set should be programmed before an MI_SET_CONTEXT command. MI_ARB_ON_OFF with 'Arbitration Enable' set should be programmed after an MI_SET_CONTEXT command.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 18h MI_SET_CONTEXT Format: OpCode
	22:8	**Reserved**
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:12	**Logical Context Address**

MI_SET_CONTEXT

		Project: CHV, BSW Format: GraphicsAddress[31:12]LogicalContext **Description** This field contains the 4KB-aligned graphics memory address of the Logical Context that is to be loaded into the hardware context. If this address is equal to the CCID register associated with the current ring, no load will occur. Prior to loading this new context, the device will save the existing context as required. After the context switch operation completes, this address will be loaded into the associated CCID register. This field needs to be 4KB aligned virtual address.
	11:10	**Reserved** Format: MBZ
	9	**Reserved**
	8	**Reserved, Must be 1** Format: Must Be One
	7:5	**Reserved**
	4	**Core Mode Enable** Project: CHV, BSW Format: Enable If set the Context Image will be offset based off the Core ID: If Core ID 0, no offset If Core ID 1, 36KB Offset
	3	**Resource Streamer State Save Enable** Project: CHV, BSW Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is saved as part of switching away from this logical context. This bit will be stored in the associated CCID register to control the context save operation when switching away from this context (as part of a subsequent MI_SET_CONTEXT command).
	2	**Resource Streamer State Restore Enable** Project: CHV, BSW Format: Enable If set, the resource streamer state identified in the Logical Context Data section of the Memory Data Formats chapter is loaded (or restored) as part of switching to this logical context. This bit affects the switch (if required) to the context specified in Logical Context Address. This bit will also be stored in the associated CCID register to control a subsequent context save operation when switching to this context (as part of a subsequent ring buffer switch).
	1	**Force Restore**

MI_SET_CONTEXT		
		When switching to this logical context a comparison between Logical Context Address and the contests of the CCID register is performed. Normally, matching addresses prevent a context restore from occurring; however, when this bit is set a context restore is forced to occur. This bit cannot be set with Restore Inhibit. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.
	0	**Restore Inhibit** If set, the restore of the HW context from the logical context specified by Logical Context Address is inhibited (i.e., the existing HW context values are maintained). This bit must be used to prevent the loading of an uninitialized logical context. If clear, the context switch proceeds normally. This bit cannot be set with Force Restore. Note: This bit is not saved in the associated CCID register. It only affects the processing of this command.

MI_SET_PREDICATE

MI_SET_PREDICATE

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

Description

This command sets the Predication Check for the subsequent commands in the command buffer except for MI_SET_PREDICATE itself. Render Command Streamer NOOPs the following commands based on the PREDICATE_ENABLE from MI_SET_PREDICATE, MI_SET_PREDICATE_RESULT and MI_SET_PREDICATE_RESULT_2 status. Resource Streamer doesn't take any action of parsing MI_SET_PREDICATE, this command is similar to any other command which is not meant for resource streamer.

Executing MI_SET_PREDICATE command sets PREDICATE_ENABLE bits in MI_MODE register, MI_MODE register gets render context save restored.

Programming Notes

- MI_SET_PREDICATE predication scope must be confined within a Batch Buffer to set of commands.
- MI_SET_PREDICATE with Predicate Enable Must always have a corresponding MI_SET_PREDICATE with Predicate Disable within the same Batch Buffer.
- MI_ARB_CHK command must be programmed outside the Predication Scope of MI_SET_PREDICATE.
- MI_SET_PREDICATE Predication Scope must not involve any RC6 triggering events.

Only the following command(s) can be programmed between the MI_SET_PREDICATE command enabled for predication: 3DSTATE_URB_VS 3DSTATE_URB_HS 3DSTATE_URB_DS 3DSTATE_URB_GS 3DSTATE_PUSH_CONSTANT_ALLOC_VS 3DSTATE_PUSH_CONSTANT_ALLOC_HS 3DSTATE_PUSH_CONSTANT_ALLOC_DS 3DSTATE_PUSH_CONSTANT_ALLOC_GS 3DSTATE_PUSH_CONSTANT_ALLOC_PS MI_LOAD_REGISTER_IMM MEDIA_VFE_STATE MEDIA_OBJECT MEDIA_OBJJECT_WALKER MEDIA_INTERFACE_DESCRIPTOR_LOAD 3DSTATE_WM_HZ_OP

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 01h MI_SET_PREDICATE Format: OpCode
	22:4	**Reserved**
	3:0	**PREDICATE ENABLE** Project: CHV, BSW This field sets the predication logic in render command streamer when parsed. Predicate Disable is the default mode of operation.

MI_SET_PREDICATE

Value	Name	Description
0h	NOOP Never	Predication is Disabled and RCS will process commands as usual.
1h	NOOP on Result2 clear	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is clear.
2h	NOOP on Result2 set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is set.
3h	NOOP on Result clear	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is clear.
4h	NOOP on Result set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is set.
5h	Execute when one slice enabled.	Following Commands will be Executed by RCS only when one slice is enabled.
6h	Execute when two slices are enabled.	Following Commands will be Executed by RCS only when two slices are enabled.
7h	Execute when three slices are enabled.	Following Commands will be Executed by RCS only when all the three slices are enabled.
8h-Ah	Reserved	
Bh, Ch	Reserved	
Dh, Eh	Reserved	
Fh	NOOP Always	Following Commands will be NOOPED by RCS unconditionally.

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The MI_STORE_DATA_IMM command requests a write of the QWord or DWord constant supplied in the packet to the specified Memory Address. This command also supports writing to consecutive dword or qword memory locations form the starting address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).

This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

This command should not be used within a non_privilege batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Format: U1 If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch or ring buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.
	21	**Store Qword** Format: U1 Value / Name / Description: 0h / Store Dword / If set, this command generates dword writes to memory. Number of dwords generated depends upon the number of 'Data Dword' programmed in the command. If 'x' number of data dwords are programmed in the command it results in 'x' dword writes to memory.

MI_STORE_DATA_IMM

		1h — Store Qword: If set, this command generates Qword writes to memory, two 'Data Dword' are paired to form a Qword. Number of qwords generated depends upon the number of 'Data Dword' programmed in the command. If 'x' number of data dwords are programmed in the command it results in 'x/2' qword writes to memory.
	20:10	**Reserved** Format: MBZ
	9:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1..2	63:2	**Destination Address** Format: GraphicsAddress[63:2] This field specifies the 4GB aligned base address within the host's 64-bit virtual address space. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned if "Store Qword" is enabled. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved** Format: MBZ
3	31:0	**Data DWord 0** Format: U32 FormatDesc This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 2

The MI_STORE_DATA_IMM command requests a write of the QWord or DWord constant supplied in the packet to the specified Memory Address. This command also supports writing to consecutive dword or qword memory locations form the starting address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

This command should not be used within a "non_privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or "privilege" batch buffers. If used within a non-privilege batch buffer, **Use Global GTT** must be clear. This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers). This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Project: All Format: U1 If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.
	21	**Store Qword** Value: 0h, Name: Store Dword, Description: If set, this command generates dword writes to memory. Number of dwords generated depends upon the number of 'Data Dword' programmed in the command. If 'x' number of data dwords are programmed in the command it results in 'x' dword writes to memory.

MI_STORE_DATA_IMM

		1h	Store Qword	If set, this command generates Qword writes to memory, two 'Data Dword' are paired to form a Qword. Number of qwords generated depends upon the number of 'Data Dword' programmed in the command. If 'x' number of data dwords are programmed in the command it results in 'x/2' qword writes to memory.
	20:10	**Reserved**		
		Project:	All	
		Format:	MBZ	
	9:0	**DWord Length**		
		Default Value:	0h Excludes DWord (0,1)	
		Format:	=n Total Length - 2	
1	31:2	**Address**		
		Format:	GraphicsAddress[31:2]U32(2)	
		This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.		
	1:0	**Reserved**		
		Format:	MBZ	
2	31:16	**Reserved**		
		Format:	MBZ	
	15:0	**Address High**		
		Format:	GraphicsAddress[47:32]U16	
		This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.		
3	31:0	**Data DWord 0**		
		Format:	U32	
		This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).		
4	31:0	**Data DWord 1**		
		Format:	U32	
		This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).		

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

Description

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

This command supports writing to multiple consecutive dword or qword memory locations from the starting address.

Programming Notes

This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers). However, the cacheable nature of the transaction is determined by the setting of the "mapping type" in the GTT entry. This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations. All writes to memory generated using this command are expected to finish in order.

This command should not be used within a "non_privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM
	22	**Use Global GTT** Project: All This bit must be '1' if the Per Process GTT Enable bit is clear. Value / Name / Description: 0h — Per Process Graphics Address 1h — Global Graphics Address — This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_STORE_DATA_IMM

	21	Store Qword **Value** / **Name** / **Description** 0h / Store Dword / If set, this command generates dword writes to memory. Number of dwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of data dwords are programmed in the command it results in "x" dword writes to memory. 1h / Store Qword / If set, this command generates Qword writes to memory, two "Data Dword" are paired to form a Qword. Number of qwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of data dwords are programmed in the command it results in "x/2" qword writes to memory.
	20:10	**Reserved**
	9:0	**DWord Length** Default Value: 2h Excludes DWord (0,1) = 2 for DWord, 3 for QWord Format: =n Total Length - 2
1 **Project:** CHV, BSW	31:2	**Address** Project: CHV, BSW Format: GraphicsAddress[31:2]U32(2) This field specifies Bits 31:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.
	1	**Reserved**
	0	**Core Mode Enable** Project: CHV, BSW Format: U1 This bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data(32b or 64b based off number of DW length).
2 **Project:** CHV, BSW	31:16	**Reserved**
	15:0	**Address High** Project: CHV, BSW Format: GraphicsAddress[47:32]U32(2) This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space.
3	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).

MI_STORE_DATA_IMM		
4	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

Description

The MI_STORE_DATA_IMM command requests a write of the QWord constant supplied in the packet to the specified Memory Address. As the write targets a System Memory Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

This command supports writing to multiple consecutive dwords or qwords memory locations from the starting address.

Programming Notes

- This command should not be used within a "non-privilege" batch buffer to access global virtual space, doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or privilege batch buffers to access global virtual space.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll un-cached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

Number of consecutive dwords or qwords programmed must be restricted such that the DWord Length doesn't exceed 0x3FE, i.e single command supports updating 1021 consecutive dword locations or 510 qword locations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 20h MI_STORE_DATA_IMM Format: OpCode
	22	**Use Global GTT** Project: All Format: Boolean

MI_STORE_DATA_IMM

		Description If set, this command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer. If clear, the PPGTT will be used. It is allowed for this bit to be clear when executing this command from a privileged (secure) batch buffer. This bit must be '1' if the Per Process GTT Enable bit is clear.
	21	**Store Qword** Project: CHV, BSW Format: Boolean If set, this command generates Qword writes to memory, two "Data Dword" are paired to form a Qword. Number of qwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x/2" qword writes to memory. If reset this command generates Dwords writes to memory. Number of dwords generated depends upon the number of "Data Dword" programmed in the command. If 'x' number of "Data Dwords" are programmed in this command it results in "x" dword writes to memory.
	20:10	**Reserved**
	9:0	**DWord Length** Project: CHV, BSW Format: =n Total Length - 2. Excludes DWord (0,1) **Value** / **Name** 2h — Store Dword **[Default]** 3h — Store Qword **Programming Notes** DWord Length programmed must not exceed 0x3FE. If RS is enabled in the batch buffer, then the value of this field must not exceed 0x3F.
1..2 **Project:** CHV, BSW	63:48	**Reserved**
	47:2	**Address** Format: GraphicsAddress[47:2] This field specifies Bits 47:2 of the Address where the DWord will be stored. As the store address must be DWord-aligned, Bits 1:0 of that address MBZ. This address must be 8B aligned for a store "QW" command.
	1	**Reserved**
	0	**Core Mode Enable** Project: CHV, BSW Format: U1 This bit is set then the address will be offset by the Core ID: If Core ID 0, then there is no offset If Core ID 1, then the Memory is offset by the size of the data(32b or 64b based off number of DW length).

MI_STORE_DATA_IMM		
3	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
4	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Project: CHV, BSW

Source: VideoEnhancementCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Project: All Format: MBZ
	21	**Use Per-Process Hardware Status Page** Project: CHV, BSW If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Project: All Format: MBZ

MI_STORE_DATA_INDEX

	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = 2 for QWord Project: All Format: =n Total Length - 2
1	31:12	**Reserved** Project: All Format: MBZ
	11:2	**Offset** Project: All Format: U10 Zero-based DWord offset into the HW status page Format: GraphicsAddress[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED. For a QWord write, the offset is valid down to bit 3 only. **Value** / **Name** [16, 1023]
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord 0** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).
3	31:0	**Data Word 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED. This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers). This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX
	22	**Reserved**
	21	**Use Per-Process Hardware Status Page** Project: CHV, BSW If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 1h Excludes DWord (0,1) = 1 for DWord, 2 for QWord Format: =n Total Length - 2
1	31:12	**Reserved** Project: All Format: MBZ

MI_STORE_DATA_INDEX

	11:2	**Offset** Project: All Format: U10 zero-based DWord offset into the HW status page. Format: HardwareStatusPageOffset[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED.This address must be 8B aligned for a store "QW" command. **Value**: [16, 1023] **Name**:
	1:0	**Reserved** Project: All Format: MBZ
2	31:0	**Data DWord 0** Project: All Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Project: All Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Project: CHV, BSW

Source: RenderCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete eventually, there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Project: CHV, BSW
	21	**Use Per-Process Hardware Status Page** Project: CHV, BSW If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be 0 if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 1h Format: =n Total Length - 2. Excludes DWord (0,1) = 1 for DWord, 2 for QWord.

MI_STORE_DATA_INDEX

1	31:12	**Reserved** Format: MBZ
	11:2	**Offset** Format: U10 zero-based DWord offset into the HW status page. Format: HardwareStatusPageOffset[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. Note that the first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED. This address must be 8B aligned for a store QW command. Value: [16, 1023] — Name:
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 This field specifies the DWord value to be written to the targeted location.For a QWord write this DWord is the lower DWord of the QWord to be reported (DW 0).
3	31:0	**Data DWord 1** Format: U32 This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_DATA_INDEX

MI_STORE_DATA_INDEX

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The MI_STORE_DATA_INDEX command requests a write of the data constant supplied in the packet to the specified offset from the System Address defined by the Hardware Status Page Address Register. As the write targets a System Address, the write operation is coherent with the CPU cache (i.e., the processor cache is snooped).

Programming Notes

- Use of this command with an invalid or uninitialized value in the Hardware Status Page Address Register is UNDEFINED.
- This command can be used for general software synchronization through variables in cacheable memory (i.e., where software does not need to poll uncached memory or device registers).
- This command simply initiates the write operation with command execution proceeding normally. Although the write operation is guaranteed to complete "eventually", there is no mechanism to synchronize command execution with the completion (or even initiation) of these operations.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 21h MI_STORE_DATA_INDEX Format: OpCode
	22	**Reserved** Format: MBZ
	21	**Use Per-Process Hardware Status Page** Project: CHV, BSW If this bit is set, this command will index into the per-process hardware status page at offset 0K from the LRCA. If clear, the Global Hardware Status Page will be indexed. This bit must be '0' if the Execlist Enable bit is clear.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) = 2 for QWord Format: =n Total Length - 2

MI_STORE_DATA_INDEX

1	31:12	**Reserved** Format: MBZ
	11:2	**Offset** Format: U10 zero-based DWord offset into the HW status page Format: GraphicsAddress[11:2]U32 This field specifies the offset (into the hardware status page) to which the data will be written. For a QWord write, the offset is valid down to bit 3 only. **Value** [16, 1023] — **Name** **Programming Notes** The first few DWords of this status page are reserved for special-purpose data storage - targeting these reserved locations via this command is UNDEFINED.
	1:0	**Reserved** Format: MBZ
2	31:0	**Data DWord 0** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).
3	31:0	**Data Word 1** Format: U32 FormatDesc This field specifies the upper DWord value to be written to the targeted QWord location (DW 1).

MI_STORE_REGISTER_MEM

MI_STORE_REGISTER_MEM

Project: CHV, BSW

Source: CommandStreamer

Length Bias: 2

The MI_STORE_REGISTER_MEM command requests a register read from a specified memory mapped register location in the device and store of that DWord to memory. The register address is specified along with the command to perform the read.

Programming Notes	Source
• The command temporarily halts command execution. • The memory address for the write is snooped on the host bus. • This command should not be used from within a "non-privilege" batch buffer to access global virtual space. doing so will be treated as privilege access violation. Refer "User Mode Privilege Command" in MI_BATCH_BUFFER_START command section to know HW behavior on encountering privilege access violation. This command can be used within ring buffers and/or "privilege" batch buffers to access global virtual space. • This command will cause undefined data to be written to memory if given register addresses for the PGTBL_CTL_0 or FENCE registers.	
Source: BlitterCS, VideoCS, VideoEnhancementCS The source MMIO offset must be limited to any MMIO that is not replicated due to multiple slice configurations. If slice zero is disabled, then any MMIO read from this command streamer to a register replicated in the slice will cause a return value of zero.	BlitterCS, VideoCS, VideoEnhancementCS

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 24h MI_STORE_REGISTER_MEM Format: OpCode
	22	**Use Global GTT** Format: Boolean It is allowed for this bit to be set when executing this command from a privileged (secure) batch or ring buffer. This bit must be clear when programmed from within a non-privileged batch buffer. This bit must be 1 if the Per Process GTT Enable bit is clear. This command will use the global GTT to translate the Address and this command must be executing from a privileged (secure) batch buffer.

MI_STORE_REGISTER_MEM

	21	**Reserved**
	21	**Predicate Enable** Project: CHV, BSW Source: RenderCS Format: U1 If set, this command is executed (or not) depending on the current value of the MI Predicate internal state bit. This command is ignored only if PredicateEnable is set and the Predicate state bit is 0.
	20:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2 Value: 2h — Name: Excludes DWord (0,1) **[Default]** — Project: CHV, BSW
1	31:23	**Reserved** Format: MBZ
	22:2	**Register Address** Format: MMIOAddress[22:2]MMIO_Register This field specifies Bits 22:2 of the Register offset the DWord will be read from. As the register address must be DWord-aligned, Bits 1:0 of that address MBZ. **Programming Notes** • Storing a VGA register is not permitted and will store an UNDEFINED value. • The values of PGTBL_CTL0 or any of the FENCE registers cannot be stored to memory; UNDEFINED values will be written to memory if the addresses of these registers are specified.
	1:0	**Reserved**
2..3 **Project:** CHV, BSW	63:2	**Memory Address** Project: CHV, BSW Format: GraphicsAddress[63:2]MMIO This field specifies the address of the memory location where the register value specified in the DWord above will be written. The address specifies the DWord location of the data.Range = GraphicsVirtualAddress[63:2] for a DWord register GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	1:0	**Reserved**

MI_STORE_URB_MEM

MI_STORE_URB_MEM

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_STORE_URB_MEM command requests a URB read from a specified memory mapped URB location in the device and store of that DWord to memory. The URB address is specified along with the command to perform the read.

Programming Notes

- The command temporarily halts command execution.
- This command should not be used within a "non-secure" batch buffer to access global virtual space. Doing so will cause the command parser to perform the write with byte enables turned off. This command can be used within ring buffers and/or "secure" batch buffers.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 2Dh MI_STORE_URB_MEM Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Format: =n Total Length - 2. Excludes DWord (0,1). Value: 2h — Name: **[Default]** — Project: CHV, BSW
1	31:15	**Reserved** Format: MBZ
	14:2	**URB Address** This field specifies Bits 14:2 of the URB offset the DWord will be read in the URB. This command only supports reading from the lower 32KB of the URB space.
	1:0	**Reserved** Format: MBZ

<table>
<tr><th colspan="3">MI_STORE_URB_MEM</th></tr>
<tr><td rowspan="2">2..3
Project:
CHV, BSW</td><td>63:6</td><td>Memory Address<table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>GraphicsAddress[63:6]</td></tr></table>This field specifies the address of the location of where the value will be written to memory. The value must be in the first DW location of the cache line. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].</td></tr>
<tr><td>5:0</td><td>Reserved<table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>MBZ</td></tr></table></td></tr>
</table>

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Project: CHV, BSW
Source: VideoEnhancementCS
Length Bias: 1

Description	Project
Blocks PM Flush Requests.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Project: All Format: MBZ
	0	**Suspend Flush** Project: All Format: Enable Description: This field suspends flush due to a PM flush request. (Project: CHV, BSW)

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Project: CHV, BSW
Source: BlitterCS
Length Bias: 1

Description	Project
Blocks PM Flush Requests.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Project: All Format: MBZ
	0	**Suspend Flush** Project: All Format: Enable Description: This field suspends flush due to a PM flush request. (Project: CHV, BSW)

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

Description	Project
Blocks PM Flush Requests.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH Format: OpCode
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable Description: This field suspends flush due to a PM flush request. (Project: CHV, BSW)

MI_SUSPEND_FLUSH

MI_SUSPEND_FLUSH

Project: CHV, BSW
Source: VideoCS
Length Bias: 1

Description	Project
Blocks PM Flush Requests.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 0Bh MI_SUSPEND_FLUSH
	22:1	**Reserved** Format: MBZ
	0	**Suspend Flush** Format: Enable Description: This field suspends flush due to a PM flush request. (Project: CHV, BSW)

MI_TOPOLOGY_FILTER

MI_TOPOLOGY_FILTER

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

This command is used to specify a specific 3DPrimType value, where the CS will ignore all 3DPRIMITIVE commands that do no have a matching 3DPrimType. This primitive culling is optional (turned off by using this command with a Topology Filter Value of 0). **This command is specific to the Render command stream only.**

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 0Dh MI_TOPOLOGY_FILTER Format: OpCode
	22:6	**Reserved** Format: MBZ
	5:0	**Topology Filter Value** Format: 3D_Prim_Topo_Type [CHV, BSW] When non-zero, the CS will discard all 3DPRIMITIVE commands which do not match the specified 3DPrimTopologyType. When zero, no filtering is performed (normal operation).

MI_UPDATE_GTT

MI_UPDATE_GTT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The MI_UPDATE_GTT command is used to update GGTT page table entries in a coherent manner and at a predictable place in the command flow.

A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries.

MI_FLUSH_DWORD flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).

MI_UPDATE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.

PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved**
	9:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**Entry Address** Format: GraphicsAddress[31:12] This field holds the QW offset of the first table entry to be modified in GGTT.
	11:0	**Reserved**
2..n	31:0	**Entry Data** Format: PageTableEntry This Dword becomes the lower dword new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

MI_UPDATE_GTT

MI_UPDATE_GTT

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The MI_UPDATE_GTT command is used to update GTT page table entries in a coherent manner and at a predictable place in the command flow.
A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A PIPE_CONTROL flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries.
PIPE_CONTROL flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).
MI_UPDTE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.
PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved**
	9:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1). **Programming Notes** The value of this field must not exceed a value 3Fh when programmed in a batch buffer with resource streamer enabled.
1	31:12	**Entry Address** Format: GraphicsAddress[31:12] This field holds the QW offset of the first table entry to be modified in GGTT.

MI_UPDATE_GTT		
	11:0	**Reserved**
2..n	63:0	**Entry Data** Format: PageTableEntry This Dword becomes the new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

MI_UPDATE_GTT

MI_UPDATE_GTT

Project: CHV, BSW

Source: VideoCS

Length Bias: 2

The MI_UPDATE_GTT command is used to update GGTT page table entries in a coherent manner and at a predictable place in the command flow.

A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries.

MI_FLUSH_DWORD flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).

MI_UPDATE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.

PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved**
	9:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**Entry Address** Format: GraphicsAddress[31:12] This field holds the QW offset of the first table entry to be modified in GGTT.
	11:0	**Reserved**
2..n	63:0	**Entry Data** Format: PageTableEntry This Dword becomes the lower dword new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

MI_UPDATE_GTT

MI_UPDATE_GTT

Project:	CHV, BSW
Source:	VideoEnhancementCS
Length Bias:	2

The MI_UPDATE_GTT command is used to update GGTT page table entries in a coherent manner and at a predictable place in the command flow.
A MI_FLUS_DWORD flush command with "CS Stall" bit set must be programmed prior to MI_UPDATE_GTT command, since work associated with preceding commands that are still in the pipeline may be referencing GTT entries that will be changed by its execution. The flush must also invalidate TLBs and read caches that may become invalid as a result of the changed GTT entries. A MI_FLUSH_DWORD flush command with "CS Stall" bit set must be programmed post MI_UPDATE_GTT command to ensure the GGTT is updated with modified page table entries before the following workload references the modified entries.
MI_FLUSH_DWORD flush is not required if it can be guaranteed that the pipeline is free of any work that relies on changing GTT entries (such as MI_UPDATE_GTT contained in a paging DMA buffer that is doing only update/mapping activities and no rendering).
MI_UPDTE_GTT command is privilege operation and will be converted to a no-op and an error flagged if it is executed from within a non-secure batch buffer.
PPGTT updates cannot be done via **MI_UPDATE_GTT**, gfx driver will have to use MI_STORE_DATA_IMM for PPGTT inline updates.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 23h MI_UPDATE_GTT Format: OpCode
	22:10	**Reserved**
	9:0	**DWord Length** Default Value: 0h Excludes DWord (0,1) Format: =n Total Length - 2
1	31:12	**Entry Address** Format: GraphicsAddress[31:12] This field holds the QW offset of the first table entry to be modified in GGTT.
	11:0	**Reserved**
2..n	63:0	**Entry Data** Format: PageTableEntry This Dword becomes the lower dword new page table entry. See PPGTT/Global GTT Table Entries (PTEs) in Memory Interface Registers.

MI_URB_ATOMIC_ALLOC

MI_URB_ATOMIC_ALLOC

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

This command is used to specify the region in URB allocated for URB atomic value storage.
This command is specific to the Render command stream only.

Programming Notes
This command can only be sent after a flush has occurred.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 09h MI_URB_ALLOC Format: OpCode
	22:20	**Reserved** Format: MBZ
	19:12	**URB Atomic Storage Offset** Format: U8 Number of 128B Entries This field specifies the offset of a 128B granular starting address in the URB. The value of **URB Atomic Storage Offset** plus the value of the **URB Atomic Storage Size** must not exceed 256. Value: [0,255] — Name: — Description: 0-(32KB-128B)
	11:9	**Reserved** Format: MBZ
	8:0	**URB Atomic Storage Size** Format: U9 Number of 128B Entries This field specifies the size of the buffer in the URB in number of 128B entries. If this field has a value of zero then the URB Atomic allocation is disabled and will not be context save/restored. Value: [0,256] — Name: — Description: 0-32KB

MI_URB_CLEAR

MI_URB_CLEAR

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

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

Programming Notes

- The command temporarily halts command execution.
- This command is part of context save/restore. Only the last instance will be part of context.
- This command requires the 3D pipeline to be flushed before execution.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 19h MI_URB_CLEAR Format: OpCode
	22:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 0h Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:30	**Reserved**
	29:16	**URB Clear Length** Project: CHV, BSW This field specifies the number of 256b entries in the URB to be cleared to zero. Value: [0,16383] — Name:
	15	**Reserved**
	14:0	**URB Address** Project: CHV, BSW Format: URBAddress[19:5] 256b aligned This field specifies Bits 19:5 of the URB Address

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Project: CHV, BSW
Source: VideoEnhancementCS
Length Bias: 1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Project: All Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT		
Project:	CHV, BSW	
Source:	BlitterCS	
Length Bias:	1	
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT
	22:0	**Reserved** Project: All Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT

Project: CHV, BSW
Source: RenderCS
Length Bias: 1

The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_USER_INTERRUPT

MI_USER_INTERRUPT		
Project:	CHV, BSW	
Source:	VideoCS	
Length Bias:	1	
The MI_USER_INTERRUPT command is used to generate a User Interrupt condition. The parser will continue parsing after processing this command. See User Interrupt.		
DWord	**Bit**	**Description**
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 02h MI_USER_INTERRUPT Format: OpCode
	22:0	**Reserved** Format: MBZ

MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	1

The MI_WAIT_FOR_EVENT command is used to pause command stream processing until a specific event occurs or while a specific condition exists. Only one event/condition can be specified -- specifying multiple events is UNDEFINED.The effect of the wait operation depends on the source of the command. If executed from a batch buffer, the parser will halt (and suspend command arbitration) until the event/condition occurs. If executed from a ring buffer, further processing of that ring will be suspended, although command arbitration (from other rings) will continue. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.If execution of this command from a primary ring buffer causes a wait to occur, the active ring buffer will effectively give up the remainder of its time slice (required in order to enable arbitration from other primary ring buffers).

Programming Notes

[Ring Buffer Mode Of scheduling Only][Blitter CS]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information following MI_WAIT_FOR_EVENT command. This will ensure Page Directory information gets reprogrammed on exiting IDLE flow triggered on MI_WAIT_FOR_EVENT. Alternatively SW can disable IDLE flows on MI_WAIT_FOR_EVENT by setting below bits in “BCS_ECOSKPD” register. Disable GT C6 Enter Due to Blitter Waiting on Vblank Disable GT C6 Enter Due to Blitter Waiting on Scanline Disable GT C6 Enter Due to Blitter Waiting on Flip Done

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT
	22	**Reserved**
	21	**Display Pipe C Vertical Blank Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait until the next Display Pipe C "Vertical Blank" event occurs. This event is described as the start of the next Display C vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event in the Device Programming Interface chapter of MI Functions. **Programming Notes** If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.

MI_WAIT_FOR_EVENT

20 **Display Sprite C Flip Pending Wait Enable**

Project:	All
Format:	Enable

This field enables a wait for the duration of a Display Sprite C "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

Programming Notes

If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.

19 **Display Sprite C3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite C3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

18 **Display Sprite B3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite B3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

17 **Display Sprite A3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite A3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

16 **Display Sprite C2 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite C2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

15 **Display Plane C Flip Pending Wait Enable**

Project:	All
Format:	Enable

MI_WAIT_FOR_EVENT

This field enables a wait for the duration of a Display Plane C "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

Programming Notes

If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.

14 **Display Pipe C Scan Line Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait while a Display Pipe C "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe C Display Scan Line Count Range Compare Register.

Programming Notes

If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.

13:12 **Reserved**

Project:	All
Format:	MBZ

11 **Display Pipe B Vertical Blank Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait until the next Display Pipe B "Vertical Blank" event occurs. This event is described as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period.

10 **Display Sprite B Flip Pending Wait Enable**

Project:	All
Format:	Enable

This field enables a wait for the duration of a Display Sprite B "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

9 **Display Plane B Flip Pending Wait Enable**

Project:	All
Format:	Enable

This field enables a wait for the duration of a Display Plane B "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

8	**Display Pipe B Scan Line Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait while a Display Pipe B "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register. **Programming Notes** If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.
7	**Display Sprite B2 Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite B2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
6	**Display Sprite A2 Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite A2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
5:4	**Reserved** Project: All Format: MBZ
3	**Display Pipe A Vertical Blank Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait until the next Display Pipe A "Vertical Blank" event occurs. This event is described as the start of the next Display Pipe A vertical blank period. Note that this can cause a wait for up to an entire refresh period.
2	**Display Sprite A Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Sprite A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

	1	**Display Plane A Flip Pending Wait Enable** Project: All Format: Enable This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
	0	**Display Pipe A Scan Line Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait while a Display Pipe A "Scan Line" condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register. **Programming Notes** If this bit is set, then RC6 cannot occur while waiting for this event to complete. SW must disable IDLE sequence or RC6 prior to the command. IDLE sequence and RC6 must be reenabled after the command to allow the hardware to enter RC6.

MI_WAIT_FOR_EVENT

MI_WAIT_FOR_EVENT

Project:	CHV, BSW
Source:	RenderCS
Length Bias:	1

Description

The MI_WAIT_FOR_EVENT command is used to pause command stream processing **of this pipe only** until a specific event occurs or while a specific condition exists. See Wait Events/Conditions, Device Programming Interface in *MI Functions*. Only one event/condition can be specified. Specifying multiple events is UNDEFINED. Once parsed, the parser will halt (and suspend command arbitration) until the event/condition occurs. Note that if a specified condition does not exist (the condition code is inactive) at the time the parser executes this command, the parser proceeds, treating this command as a no-operation.
If CSunit is waiting for V-blank or flip done, HW can go into RC1/RC6 state.
MI_NOOP setting NOP register (or any other benign command) must be set after MI_WAIT_FOR_EVENT under the following conditions:

- Back-to-back MI_WAIT_FOR_EVENT commands
- MI_WAIT_FOR_EVENT is the last command before head = tail

Events must be unmasked in the Display Engine Render Response Mask Register (DE RRMR 0x44050) prior to waiting for them with a MI_WAIT_FOR_EVENT command, or in the case of flips or scanlines, prior to starting the flip or loading the scanline. Unmasked events will wake command streamer as they occur, so for improved power savings it is recommended to only unmask events that are required. Programming the DE RRMR register can be done through MMIO or a LOAD_REGISTER_IMMEDIATE command.

Execution List Mode of Scheduling: CS on evaluating MI_WAIT_FOR_EVENT to be unsuccessful (has to wait for event to happen) triggers synchronous context switch stating the switch reason in Context Status Buffer. Note that synchronous context switch can be inhibited through programming "**Inhibit Synchronous Context Switch**" bit in CTXT_SR_CTL register or by disabling arbitration through MI_ARB_ON_OFF command.

Programming Notes

Ring Buffer Mode of Scheduling Only: SW must always program a dummy **MI_SEMAPHORE_WAIT** command in **Signal Mode** which is always successful prior to programming MI_WAIT_FOR_EVENT.

If the above programming restriction is not followed, in certain order of programming sequences HW would enter IDLE_DOP instead of IDLE_C6 on encountering MI_WAIT_FOR_EVENT unsuccessful.

Render CS Only: SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set prior to programming MI_WAIT_FOR_EVENT command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies for certain programming sequences.
If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.

Execlist Mode of Scheduling Only: When MI_WAIT_FOR_EVENT command results in context switch on "Wait On V-blank", "Display Plane" field indicates the Display pipe for which the wait for event was un-successful.

MI_WAIT_FOR_EVENT

When a context switched out due to "Wait on V-blank" is resubmitted by scheduler without waiting for the corresponding V-blank event to be satisfied, it might happen that the context can get switched out again by HW due to V-blank not satisfied, when this condition occurs due to known HW issue "Display Plane" field is not indicated correctly. To work around this issue SW must do one of the below

1. Always program MI_WAIT_FOR_EVENT for V-blank with Inhibit Synchronous Context Switch Set so that a context never gets switched out with context switch reason as only due to “Wait On V-blank”. However a context waiting for V-blank event to be satisfied can get switched out due to preemption on submission of a pending execlist. In this case scheduler can resubmit the switched out context at appropriate time similar to any other preempted context. **OR**
2. When MI_WAIT_FOR_EVENT for V-blank is programmed with Inhibit Synchronous Context Switch Reset. Scheduler on detecting a context switched out due to Wait On V-blank must not resubmit the context unless the corresponding V-blank event is satisfied. **OR**
3. Scheduler must maintain its own record of the Pipe assigned to a given context for Wait on V-blank and must not look at the "Display Plane" field reported in the context swtich status.

Execlist Mode of Scheduling: CS takes couple of clocks to parse MI_WAIT_FOR_EVENT command and in a specific clock while parsing it makes the decision to do Synchronous Context Switch if the event is unsuccessful. When CS receives Display message satisfying the MI_WAIT_FOR_EVENT on the same clock when it makes decision to do synchronous context switch. Only way this issue can be avoided is by inhibiting synchronous context switch on MI_WAIT_FOR_EVENT command. Synchronous context switch on MI_WAIT_FOR_EVENT can be inhibited by setting "**Inhibit Synchronous Context Switch**" bit in CTXT_SR_CTL register through MI_LOAD_REGISTER_IMM command prior to MI_WAIT_FOR_EVENT command and resetting the same after MI_WAIT_FOR_EVENT command.

Ex:

MI_LOAD_REGISTER_IMM (CTXT_SR_CTL, Inhibit Synchronous Context Switch set)
MI_WAIT_FOR_EVENT
MI_LOAD_REGISTER_IMM (CTXT_SR_CTL, Inhibit Synchronous Context Switch reset)

When the above sequence has to be programmed in a non-privileged batch buffer, SW has to ensure CTXT_SR_CTL register is forced to non-privileged register through one of the FORCE_TO_NONPRIV registers.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 0h MI_COMMAND Format: OpCode
	28:23	**MI Command Opcode** Default Value: 03h MI_WAIT_FOR_EVENT Format: OpCode
	22	**Reserved**

MI_WAIT_FOR_EVENT

21 **Display Pipe C Vertical Blank Wait Enable**

Project:	CHV, BSW
Format:	Enable

Description
This field enables a wait until the next Display Pipe C Vertical Blank event occurs. This event is described as the start of the next Display C vertical blank period. Note that this can cause a wait for up to an entire refresh period.
Render and Blitter Engines

20 **Display Sprite C Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

Description
This field enables a wait for the duration of a Display Sprite C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
Render and Blitter Engines

19 **Display Sprite C3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite C3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

18 **Display Sprite B3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite B3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

17 **Display Sprite A3 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite A3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

MI_WAIT_FOR_EVENT

16 **Display Sprite C2 Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite C2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

15 **Display Plane C Flip Pending Wait Enable**

Project:	CHV, BSW
Format:	Enable

Description
This field enables a wait for the duration of a Display Plane C "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
Render and Blitter Engines

14 **Display Pipe C Scan Line Wait Enable**

Project:	CHV, BSW
Format:	Enable

Description
This field enables a wait while a Display Pipe C Scan Line condition exists. This condition is defined as the start of the scan line specified in the Pipe C Display Scan Line Count Range Compare Register.
Render and Blitter Engines

13 **Reserved**

Project:	CHV, BSW
Format:	MBZ

12 **Reserved**

11 **Display Pipe B Vertical Blank Wait Enable**

Format:	Enable

Description
This field enables a wait until the next Display Pipe B "Vertical Blank" event occurs. This event is described as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period.
Render and Blitter Engines

MI_WAIT_FOR_EVENT

10 Display Sprite B Flip Pending Wait Enable

Format:	Enable

Description
This field enables a wait for the duration of a Display Sprite B "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).
Render and Blitter Engines

9 Display Plane B Flip Pending Wait Enable

Format:	Enable

Description
This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers.
Render and Blitter Engines

8 Display Pipe B Scan Line Wait Enable

Format:	Enable

Description
This field enables a wait while a Display Pipe B Scan Line condition exists. This condition is defined as the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register.
Render and Blitter Engines

7 Display Sprite B2 Flip Pending Wait Enable

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite B2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

6 Display Sprite A2 Flip Pending Wait Enable

Project:	CHV, BSW
Format:	Enable

This field enables a wait for the duration of a Display Sprite A2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers).

5 Reserved

4 Reserved

MI_WAIT_FOR_EVENT

	3	**Display Pipe A Vertical Blank Wait Enable** Format: Enable **Description** This field enables a wait until the next Display Pipe A "Vertical Blank" event occurs. This event is described as the start of the next Display Pipe A vertical blank period. Note that this can cause a wait for up to an entire refresh period. **Render and Blitter Engines**
	2	**Display Sprite A Flip Pending Wait Enable** Format: Enable **Description** This field enables a wait for the duration of a Display Sprite A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). **Render and Blitter Engines**
	1	**Display Plane A Flip Pending Wait Enable** Format: Enable **Description** This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). **Render and Blitter Engines**
	0	**Display Pipe A Scan Line Wait Enable** Format: Enable **Description** This field enables a wait while a Display Pipe A "Scan Line" condition exists. This condition is defined as the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register. **Render and Blitter Engines**

Move

mov - Move

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The mov instruction moves the components in src0 into the channels of dst. If src0 and dst are of different types, format conversion is performed. If src0 is a scalar immediate, the immediate value is loaded into enabled channels of dst. A mov with the same source and destination type, no source modifier, and no saturation is a raw move. A packed byte destination region (B or UB type with HorzStride == 1 and ExecSize > 1) can only be written using raw move.

When denorm mode is flush to zero, a raw mov instruction with saturation modifier will not flush the denorm input or output to zero (Denorm is preserved).

Format: [(pred)] mov[.cmod] (exec_size) dst src0

Programming Notes

A *mov* instruction with a source modifier always copies a denorm source value to a denorm destination value (in the manner of a raw move).

There is no direct conversion from B/UB to DF or DF to B/UB. Use two instructions and a word or DWord intermediate type.

There is no direct conversion from B/UB to Q/UQ or Q/UQ to B/UB. Use two instructions and a word or DWord intermediate integer type.

There is no direct conversion from HF to DF or DF to HF. Use two instructions and F (Float) as an intermediate type.

There is no direct conversion from HF to Q/UQ or Q/UQ to HF. Use two instructions and F (Float) or a word integer type or a DWord integer type as an intermediate type.

Restriction

Raw move is not supported for Float values in ALT mode if any values are infinities or NaNs.

An accumulator can be a source or destination operand but not both.

Syntax

[(pred)] mov[.cmod] (exec_size) reg reg [(pred)] mov[.cmod] (exec_size) reg imm32 [(pred)] mov[.cmod] (exec_size) reg imm64

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

mov - Move

Src Types	Dst Types	Project
*B,*W,*D	*B,*W,*D	
*B,*W,*D	F	
F	*B,*W,*D	
F	F	
*W,*D	DF	CHV, BSW
F	DF	CHV, BSW
DF	*W,*D	CHV, BSW
DF	F	CHV, BSW
DF	DF	CHV, BSW
*W,*D,*Q	*W,*D,*Q	CHV, BSW
F	*Q	CHV, BSW
DF	*Q	CHV, BSW
*Q	F	CHV, BSW
*Q	DF	CHV, BSW
*B,*W,*D	HF	CHV, BSW
F	HF	CHV, BSW
HF	*B,*W,*D	CHV, BSW
HF	F	CHV, BSW
HF	HF	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Move Indexed

movi - Move Indexed

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The movi instruction performs a fast component-wise indexed move for subfields from src0 to dst. The source operand must be an indirectly-addressed register. All channels of the source operand share the same register number, which is provided by the register field of the first address subregister, with a possible immediate register offset. The register fields of the subsequent address subregisters are ignored by hardware. The subregister number of a source channel is provided by the subregister field of the corresponding address subregister, with a possible immediate subregister offset.
The destination register may be either a directly-addressed or an indirectly-addressed register.
This instruction effectively performs a subfield shuffling from one register to another. Up to eight subfields can be selected by an instruction.

Format: [(pred)] movi (exec_size) dst src0 src1

Programming Notes

HW Implementation Details:
The source register is calculated by adding the register portion of the first index register with the register portion of the address immediate, a0.0[11:5] + addr_imm[9:5]
For byte movi, byte0 of the destination is selected by (a0.0[4:0]), byte1 is selected by (a0.1[4:0]), ..., and byte7 is selected by (a0.7[4:0]). The rest of the bytes are undefined.
For word movi, byte0 of the destination is selected by (a0.0[4:1] & 0), byte1 is selected by (a0.0[4:1] & 1), byte2 is selected by (a0.1[4:1] & 0), byte3 is selected by (a0.1[4:1] & 1), ..., and byte15 is selected by (a0.7[4:1] & 1). The rest of the bytes are undefined.
For DWord or float movi, byte0 of the destination is selected by (a0.0[4:2] & 00b), byte1 is selected by (a0.0[4:2] & 01b), byte2 is selected by (a0.0[4:2] & 10b), byte3 is selected by (a0.0[4:2] & 11b), byte4 is selected by (a0.1[4:2] & 00b), byte5 is selected by (a0.1[4:2] & 01b), ..., byte31 is selected by (a0.7[4:2] & 11b).
For all 3 conditions above, a0.n[4:0] = a0.n[4:0] + addr_imm[4:0].

Restriction

- Source operand cannot be accumulators. The source operand must be a general register.
- The source and destination must have the same type.
- The execution size must be 8.
- The address register for the source must be aligned to the base (a0.0).
- The destination register (directly or indirectly addressed) must be 16-byte aligned.
- The destination region (directly or indirectly addressed) must point to the same GRF register.
- The destination stride in bytes must equal the source element size in bytes.
- The Align16 access mode is not allowed.
- All the index registers (address subregisters) used must point to the same GRF register.
- The instruction must use 1x1 indirect regioning.

movi - Move Indexed

The destination offset is only used to create channel enables. Each element of the destination is directly mapped to the index registers for the movi instruction. i.e. a0.0 -> dst.0, a0.1 -> dst.1, a0.2 -> dst.2, etc.

Only 8 address subregisters are used (a0.0-a0.7). Destination element 8 will be sourced from address register zero (a0.0), dst.9 <-a0.1, etc. This is an exception to the above restriction, for example:
movi (8) r31.8:uw r[a0.0,0]<8;8,1>:uw // r31.8:uw<-a0.0:uw, r31.9:uw<-a0.1:uw, etc.

Conditional Modifier is not allowed for this instruction.

Syntax

[(pred)] movi (exec_size) reg reg imm

Pseudocode

Evaluate(WrEn); srcregfile = regfile(src0); srcregbase = reg(address[0]) + reg(addr_imm); for (n = 0; n < RegWidth; n++) { if (WrEn.chan[n]) { srcsubreg = subreg(address[n] + addr_imm); dst.chan[n] = srcregfile.srcreg.srcsubreg; } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
B	B
UB	UB
W	W
UW	UW
D	D
UD	UD
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Multiply

mul - Multiply

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The mul instruction performs component-wise multiplication of src0 and src1 and stores the results in dst. When multiplying integer datatypes, if src0 is DW and src1 is W, irrespective of the destination datatype, the accumulator maintains full 48-bit precision. This is required to handle the macro for 32x32 multiplication. The macro described in the mach instruction should be used to obtain the full precision 64-bit multiplication results. Note: A 32x32 multiply operation is handled natively, without a macro. When operating in this mode, the resulting 64-bit data is packed, unlike the macro, where the lower and upper 32 bits of the result are written to different general registers by two separate instructions. Refer to the macro description for details. When multiplying integer data types, if one of the sources is a DW, the resulting full precision data is stored in the accumulator. However, if the destination data type is either W or DW, the low bits of the result are written to the destination register and the remaining high bits are discarded. This results in undefined Overflow and Sign flags. Therefore, conditional modifiers and saturation (.sat) cannot be used in this case.

Format: [(pred)] mul[.cmod] (exec_size) dst src0 src1

Restriction

Integer source operands cannot be accumulators.

When multiplying a DW and any lower precision integer, the DW operand must on src0.

[CHV, BSW-A]: DW * W is not supported outside MACH macro.

[CHV, BSW]: When multiplying DW x DW, the dst cannot be accumulator.

Syntax

[(pred)] mul[.cmod] (exec_size) reg reg reg [(pred)] mul[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n] * src1.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
*B	*B	
*B	*W	
*B	*D	
*W	*W	
*W	*D	
*W,*D	*D	

mul - Multiply

*D	*Q	
F	F	
DF	DF	
HF	HF	
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Multiply Accumulate

mac - Multiply Accumulate

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The mac instruction takes component-wise multiplication of src0 and src1, adds the results with the corresponding accumulator values, and then stores the final results in dst.

Format: [(pred)] mac[.cmod] (exec_size) dst src0 src1

Programming Notes

When source and destination datatypes are different, the implied datatype for the accumulator operand is always the destination datatype.

Restriction

Accumulator is an implicit source and thus cannot be an explicit source operand.

Syntax

[(pred)] mac[.cmod] (exec_size) reg reg reg [(pred)] mac[.cmod] (exec_size) reg reg imm32

Pseudocode

Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { dst.chan[n] = src0.chan[n] * src1.chan[n] + acc0.chan[n]; } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
*B,*W	*B,*W,*D	
F	F	
DF	DF	CHV, BSW
HF	HF	CHV, BSW
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]

mac - Multiply Accumulate		
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Multiply Accumulate High

mach - Multiply Accumulate High

Project:	CHV, BSW
Source:	EuIsa
Length Bias:	4

The mach instruction performs DWord integer multiply-accumulate operation and outputs the high DWord (bits 63:32). For each enabled channel, this instruction multiplies the DWord in src0 with the high word of the DWord in src1, left shifts the result by 16 bits, adds it with the corresponding accumulator values, and keeps the whole 64-bit result in the accumulator. It then stores the high DWord (bits 63:32) of the results in dst. This instruction is intended to be used to emulate 32-bit DWord integer multiplication by using the large number of bits available in the accumulator. For example, the following instructions perform vector multiplication of two 32-bit signed integer sources from r2 and r3 and store the resulting vectors with the high 32 bits in r5 and the low 32 bits in r6. mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d mov (8) r6.0<1>:d acc0:d // Low 32 bits. Here is a different example including negation. An added preliminary mov is required for source modification on src1. mov (8) r3.0<1>:d -r3<8;8,1>:d mul (8) acc0:d r2.0<8;8,1>:d r3.0<16;8,2>:uw mach (8) r5.0<1>:d r2.0<8;8,1>:d r3.0<8;8,1>:d // High 32 bits mov (8) r6.0<1>:d acc0:d // Low 32 bits. The mach should have channel enable from the destHI of IMUL, the mov should have the channel enable from the destLO of IMUL. As mach is used to generate part of the 64-bit DWord integer results, saturation modifier should not be used. In fact, saturation modifier should not be used for any of these four instructions. Source and destination operands must be DWord integers. Source and destination must be of the same type, signed integer or unsigned integer. If dst is UD, src0 and src1 may be UD and/or D. However, if any of src0 and src1 is D, source modifier (abs) must be present to convert it to match with dst. If dst is D, src0 and src1 must also be D. They cannot be UD as it may cause unexpected overflow because the computed results are limited to 64 bits.

Format: [(pred)] mach[.cmod] (exec_size) dst src0 src1

Restriction

Accumulator is an implicit source and thus cannot be an explicit source operand.

AccWrEn is required.

Syntax

[(pred)] mach[.cmod] (exec_size) reg reg reg [(pred)] mach[.cmod] (exec_size) reg reg imm32

Pseudocode

Evaluate(WrEn);

```
 for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { acc.chan[n][63:0] =
(src1.chan[n][31:16] * src0.chan[n][31:0]) « 16 + acc.chan[n][63:0];
dst.chan[n][31:0] = acc.chan[n][63:32]; } }
```

Errata	Description
	A source modifier must not be used on src1 for the macro operation. This applies to both mul and mach of the macro. If source modifier is required, an additional mov instruction may be used before the macro.

mach - Multiply Accumulate High

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	Y	Y

Src Types	Dst Types
D	D
UD	UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Multiply Add

mad - Multiply Add

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The mad instruction takes component-wise multiplication of src1 and src2, adds the results with the corresponding src0 values, and then stores the final results in dst.

Format: [(pred)] mad[.cmod] (exec_size) dst src0 src1 src2

Restriction	Project
[CHV, BSW]: No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.	CHV, BSW
[CHV, BSW]: All three-source instructions have certain restrictions, described in Instruction Formats [CHV, BSW].	CHV, BSW

Syntax

[(pred)] mad[.cmod] (exec_size) reg reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
F	F	
DF	DF	CHV, BSW
HF	HF	CHV, BSW
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]

mad - Multiply Add

	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved** Format: MBZ
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.

mad - Multiply Add			
	32	**Reserved**	
		Format:	MBZ
	31:0	**Header**	
		Format:	EU_INSTRUCTION_HEADER [CHV, BSW]

Multiply Add for Macro

madm - Multiply Add for Macro

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The madm instruction takes component-wise multiplication of src1 and src2, adds the results with the corresponding src0 values, and then stores the final results in dst. The source and destination operands have a higher precision carried in the exponent for this operation. The madm instruction is used for macro operations, where precision is accumulated over several instructions. This accumulation requires the exponent to increase by 2 extra bits across multiple madm operations. Refer to Macros Defined in 'Math' Section for usage and restrictions of this operation.

Format: [(pred)] madm[.cmod] (exec_size) dst src0 src1 src2

Restriction

Accumulator access is restricted to the special accumulators (acc2-acc9). Refer to the Accumulator Section for details on the special accumulators.

Syntax

[(pred)] madm[.cmod] (exec_size) reg reg reg reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src1.chan[n] * src2.chan[n] + src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

Src Types	Dst Types
F	F
DF	DF

DWord	Bit	Description
0..3	127:126	**Reserved** Format: MBZ
	125:106	**Source 2** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105	**Reserved** Format: MBZ
	104:85	**Source 1** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]

madm - Multiply Add for Macro

	84	**Reserved** Format: MBZ
	83:64	**Source 0** Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number** Format: DstSubRegNum[2:0]
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:42	**Reserved** Format: MBZ
	41:40	**Source 2 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	39:38	**Source 1 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	41:36	**Reserved** Exists If: //([Property[Source Modifier]=='false') Format: MBZ
	37:36	**Source 0 Modifier** Exists If: //([Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	35	**Reserved** Format: MBZ
	34	**Reserved** Format: MBZ
	33	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.

madm - Multiply Add for Macro

	32	**Reserved** Format: MBZ
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

No Operation

nop - No Operation

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

Do nothing. The nop instruction takes an instruction dispatch but performs no operation. It can be used for assembly patching in memory, or to insert a delay in the program sequence.

Format: nop

Restriction

The nop instruction takes no instruction options other than Breakpoint.

Syntax

```
nop
```

Pseudocode

```
{ ; // The null statement, which does nothing. }
```

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

DWord	Bit	Description
0..3	127:31	**Reserved** Format: MBZ
	30	**Reserved**
	29:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE

Oword Block Read MSD

MSD0R_OWB - Oword Block Read MSD

Project:	CHV, BSW
Source:	DataPort 0
Length Bias:	1
Family:	Block R/W
Group:	OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 00h Project: All Format: Opcode Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:11	**Reserved**
	10:8	**Data Elements** Project: All Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Block Write MSD

MSD0W_OWB - Oword Block Write MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: OW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 08h Project: All Format: Opcode Oword Block Write message
	13:11	**Reserved** Project: All Format: MBZ Ignored
	10:8	**Data Elements** Project: All Format: MDC_DB_OW Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_A32 Specifies the Binding Table Index for the message

Oword Dual Block Read MSD

MSD0R_OWDB - Oword Dual Block Read MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 02h Project: All Format: Opcode Oword Block Read message
	13	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	12:10	**Reserved**
	9:8	**Data Elements** Project: All Format: MDC_DB_OWD [CHV, BSW] Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Oword Dual Block Write MSD

MSD0W_OWDB - Oword Dual Block Write MSD

Project:	CHV, BSW
Source:	DataPort 0
Length Bias:	1
Family:	Block R/W
Group:	OW Dual Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: Enable If set, indicates that the message includes the header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 0Ah Project: All Format: Opcode Oword Block Read message
	13:10	**Reserved**
	9:8	**Data Elements** Project: All Format: MDC_DB_OWD [CHV, BSW] Specifies the number of contiguous Owords to be read or written
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Oword Unaligned Block Read MSD

MSD0R_OWUB - Oword Unaligned Block Read MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: OW Unaligned Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Legacy Message** Default Value: 0h Project: All Format: Opcode Legacy Message
	17:14	**Message Type** Default Value: 01h Project: All Format: Opcode Oword Unaligned Block Read message
	13:11	**Reserved**
	10:8	**Data Elements** Project: All Format: MDC_DB_OW [CHV, BSW] Specifies the number of contiguous Owords to be read
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_A32 [CHV, BSW] Specifies the Binding Table Index for the message

PIPE_CONTROL

PIPE_CONTROL

Project: CHV, BSW
Source: RenderCS
Length Bias: 2

The PIPE_CONTROL command is used to effect the synchronization described above.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 3h GFXPIPE_3D Format: OpCode
	26:24	**3D Command Opcode** Default Value: 2h PIPE_CONTROL Format: OpCode
	23:16	**3D Command Sub Opcode** Default Value: 0h PIPE_CONTROL Format: OpCode
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Default Value: 4h DWORD_COUNT_n Project: CHV, BSW Format: =n Total Length - 2. Excludes DWord (0,1).
1	31:29	**Reserved** Project: All Format: MBZ
	28	**Reserved** Project: CHV, BSW Format: MBZ
	27	**Reserved** Project: CHV, BSW

PIPE_CONTROL

26 Reserved

25 Reserved

Project:	All
Format:	MBZ

24 Destination Address Type

Project:	CHV, BSW

Defines address space of Destination Address

Value	Name	Description	Project
0h	PPGTT	Use PPGTT address space for DW write	All
1h	GGTT	Use GGTT address space for DW write	All

Programming Notes
Ignored if ""No Write" is selected in Operation.

23 LRI Post Sync Operation

Project:	CHV, BSW

Value	Name	Description	Project
0h	No LRI Operation	No LRI operation occurs as a result of this instruction. The Post-Sync Operation field is valid and may be used to specify an operation.	All
1h	MMIO Write Immediate Data	Write the DWord contained in Immediate Data Low (DW3) to the MMIO offset specifed in the Address field.	All

Programming Notes
This bit caues a post sync operation with an LRI (Load Register Immediate) operation. If this bit is set then the Post-Sync Operation field must be cleared.

22 Reserved

Project:	All

21 Store Data Index

Project:	
Format:	U1

Ring Buffer Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is actually an index into the global hardware status page. This bit only applies to the Global HW status page. If this field is 1, the Destination Address Type in this command must be set to 1 (GGTT). Execlist Mode Scheduling: This field is valid only if the post-sync operation is not 0. If this bit is set, the store data address is index into the global hardware status page when destination address type in the command is set to 1 (GGTT). The store data address is index into the per-process hardware status page when destination address type in the command is set to 0 (PPGTT).

PIPE_CONTROL

20 **Command Streamer Stall Enable**

Project:	All
Format:	U1

If ENABLED, the sync operation will not occur until all previous flush operations pending a completion of those previous flushes will complete, including the flush produced from this command. This enables the command to act similar to the legacy MI_FLUSH command.

Programming Notes

One of the following must also be set:

- Render Target Cache Flush Enable ([12] of DW1)
- Depth Cache Flush Enable ([0] of DW1)
- Stall at Pixel Scoreboard ([1] of DW1)
- Depth Stall ([13] of DW1)
- Post-Sync Operation ([13] of DW1)
- DC Flush Enable([5] of DW1)

This field must always be set in all PIPE_CONTROL commands having "Post-Sync Operation" set to "Report PS Depth Count" or "Report Time Stamp" when user wishes to set "Post-Sync Operation" bit in any of the MI_ATOMIC or MI_SEMAPHORE_SIGNAL commands programmed.

This bit must be always set when PIPE_CONTROL command is programmed by GPGPU and MEDIA workloads, except for the cases when only Read Only Cache Invalidation bits are set (State Cache Invalidation Enable, Instruction cache Invalidation Enable, Texture Cache Invalidation Enable, Constant Cache Invalidation Enable). This is to WA FFDOP CG issue.

19 **Reserved**

18 **TLB Invalidate**

Project:	All
Format:	U1

If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting

If ENABLED, PIPE_CONTROL command will flush the in flight data written out by render engine to Global Observation point on flush done. Also Requires stall bit ([20] of DW1) set.

Programming Notes

If ENABLED, all TLBs belonging to Render Engine will be invalidated once the flush operation is complete. Note that if the flush TLB invalidation mode is clear, a TLB invalidate will occur irrespective of this bit setting.

17 **Reserved**

Project:	CHV, BSW

PIPE_CONTROL

	Format:	MBZ

16 **Generic Media State Clear**

Project:	CHV, BSW
Format:	Disable

If set, all generic media state context information will be invalidated. Any state invalidated will not be saved as part of the render engine context image. The state only only become valid once it is parsed by the command streamer.

15:14 **Post Sync Operation**

Description
This field specifies an optional action to be taken upon completion of the synchronization operation.
This field must be cleared if the LRI Post-Sync Operation bit is set.

Value	Name	Description	Project
0h	No Write	No write occurs as a result of this instruction. This can be used to implement a "trap" operation, etc.	All
1h	Write Immediate Data	Write the QWord containing Immediate Data Low, High DWs to the Destination Address	All
2h	Write PS Depth Count	Write the 64-bit PS_DEPTH_COUNT register to the Destination Address	All
3h	Write Timestamp	Write the 64-bit TIMESTAMP register to the Destination Address	All

Programming Notes
If executed in non-secure batch buffer, the address given will be in a PPGTT address space. If in a secure ring or batch, address given will be in GGTT space

Workaround
Workaround: PIPECONTROL command with “Command Streamer Stall Enable” must be programmed prior to programming a PIPECONTROL command with Post Sync Op in GPGPU mode of operation (i.e when PIPELINE_SELECT command is set to GPGPU mode of operation).

13 **Depth Stall Enable**

Project:	All
Format:	Enable

This bit must be set when obtaining a "visible pixel" count to preclude the possible inclusion in the PS_DEPTH_COUNT value written to memory of some fraction of pixels from objects initiated after the PIPE_CONTROL command.

PIPE_CONTROL

Value	Name	Description
0h	Disable	3D pipeline will not stall subsequent primitives at the Depth Test stage.
1h	Enable	3D pipeline will stall any subsequent primitives at the Depth Test stage until the Sync and Post-Sync operations complete.

Programming Notes
This bit must be DISABLED for operations other than writing PS_DEPTH_COUNT.
This bit will have no effect (besides preventing write cache flush) if set in a PIPE_CONTROL command issued to the Media pipe.

12 **Render Target Cache Flush Enable**

Project:	All
Format:	Enable

Setting this bit will force Render Cache to be flushed to memory prior to this synchronization point completing. This bit must be set for all write fence sync operations to assure that results from operations initiated prior to this command are visible in memory once software observes this synchronization.

Value	Name	Description	Project
0h	Disable Flush	Render Target Cache is NOT flushed.	All
1h	Enable Flush	Render Target Cache is flushed.	All

Programming Notes
This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.
This bit must not be set when Depth Stall Enable bit is set in this packet.

11 **Instruction Cache Invalidate Enable**

Project:	All
Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 at the top of the pipe i.e. at the parsing time.

10 **Texture Cache Invalidation Enable**

Project:	All
Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the texture caches at the top of the pipe i.e. at the parsing time.

9 **Indirect State Pointers Disable**

Project:	All

PIPE_CONTROL

Format:	Enable

Description

At the completion of the post-sync operation associated with this pipe control packet, the indirect state pointers in the hardware are considered invalid; the indirect pointers are not saved in the context. If any new indirect state commands are executed in the command stream while the pipe control is pending, the new indirect state commands are preserved.

[CHV, BSW]: Using Invalidate State Pointer (ISP) only inhibits context restoring of Push Constant (3DSTATE_CONSTANT_*) commands. Push Constant commands are only considered as Indirect State Pointers. Once ISP is issued in a context, SW must initialize by programming push constant commands for all the shaders (at least to zero length) before attempting any rendering operation for the same context.

8 **Notify Enable**

Project:	All
Format:	Enable

If ENABLED, a Sync Completion Interrupt will be generated (if enabled by the MI Interrupt Control registers) once the sync operation is complete. See Interrupt Control Registers in Memory Interface Registers for details.

7 **Pipe Control Flush Enable**

Project:	CHV, BSW
Format:	Enable

Hardware on parsing PIPECONTROL command with Pipe Control Flush Enable set will wait for all the outstanding post sync operations corresponding to previously executed PIPECONTROL commands are complete before making forward progress.

6 **Reserved**

5 **DC Flush Enable**

Project:	CHV, BSW
Format:	Enable

Setting this bit enables flushing of the L3$ portions that caches DC writes.

Programming Notes

DC Flush (L3 Flush) by default doesn't result in flushing/invalidating the IA Coherent lines from L3$, however this can be achieved by setting control bit "**Pipe line flush Coherent lines"** in "L3SQCREG4" register.

4 **VF Cache Invalidation Enable**

Project:	All
Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of VF address based cache at the top of the pipe i.e. at the parsing time.

PIPE_CONTROL

Workaround	Project
Workaround: When VF Cache Invalidate is set “Post Sync Operation” must be enabled to “Write Immediate Data” or “Write PS Depth Count” or “Write Timestamp”.	CHV, BSW

3 **Constant Cache Invalidation Enable**

Project:	All
Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the constant cache at the top of the pipe i.e. at the parsing time.

2 **State Cache Invalidation Enable**

Project:	All
Format:	Enable

Setting this bit is independent of any other bit in this packet. This bit controls the invalidation of the L1 and L2 state caches at the top of the pipe i.e. at the parsing time.

1 **Stall At Pixel Scoreboard**

Project:	All
Format:	Enable

Defines the behavior of PIPE_CONTROL command at the pixel scoreboard.

Value	Name	Description	Project
0h	Disable	Stall at the pixel scoreboard is disabled.	All
1h	Enable	Stall at the pixel scoreboard is enabled.	All

Programming Notes
This bit must be DISABLED for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries. This bit is ignored if Depth Stall Enable is set. Further the render cache is not flushed even if Write Cache Flush Enable bit is set.

0 **Depth Cache Flush Enable**

Project:	All
Format:	Enable

Setting this bit enables flushing (i.e. writing back the dirty lines to memory and invalidating the tags) of depth related caches. This bit applies to HiZ cache, Stencil cache and depth cache.

Value	Name	Description	Project
0h	Flush Disabled	Depth relates caches (HiZ, Stencil and Depth) are NOT flushed.	All
1h	Flush Enabled	Depth relates caches (HiZ, Stencil and Depth) are flushed.	All

Programming Notes
Ideally depth caches need to be flushed only when depth is required to be coherent in memory for later use as a texture, source or honoring CPU lock. This bit must be DISABLED

PIPE_CONTROL

		for End-of-pipe (Read) fences, PS_DEPTH_COUNT or TIMESTAMP queries.
2	31:2	**Address** Project: CHV, BSW Format: GraphicsAddress[31:2]U32 If **Post Sync Operation** is set to 1h ([CHV, BSW]: **LRI Post-Sync Operation** must be clear): Bits 31:3 secify the QW address of where the Immediate Data following this DW in the packet to be stored. Bit 2 MBZ Ignored if "No Write" is the selected in Post-Sync Operation [CHV, BSW]: If **LRI Post-Sync Operation** is set: Bits 22:2 (Bits 31:23 are reserved MBZ) specify the MMIO offset destination for the data in the **Immediate Data Low** (DW3) field. Only DW writes are valid.
	1:0	**Reserved**
3 **Project:** CHV, BSW	31:16	**Reserved**
	15:0	**Address High** Project: All Format: GraphicsAddress[47:32]U32 This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space. This field is valid only if the post-sync operation is not 0 and the LRI Post-Sync Operation is clear.
4..5 **Project:** CHV, BSW	63:0	**Immediate Data** Project: CHV, BSW Format: U64 This field specifies the QWord value to be written to the targeted location. Only valid when Post-Sync Operation is 1h (Write Immediate Data) or LRI Post-Sync Operation is set. Ignored if Post-Sync Operation is "No write", "Write PS_DEPTH_COUNT" or "Write TIMESTAMP". **Programming Notes** / **Project** This field must be programmed to 0 when Post-Sync Operation is set to Write PS Depth Count or Write Timestamp. / CHV, BSW

PIPELINE_SELECT

PIPELINE_SELECT

Project:	CHV, BSW
Source:	PRM
Length Bias:	1

Description

The PIPELINE_SELECT command is used to specify which GPE pipeline is to be considered the 'current' active pipeline. Issuing 3D-pipeline-specific commands when the Media pipeline is selected, or vice versa, is UNDEFINED.

Issuing 3D-pipeline-specific commands when the GPGPU pipeline is selected, or vice versa, is UNDEFINED.

Programming common non pipeline commands (e.g., STATE_BASE_ADDRESS) is allowed in all pipeline modes.

Programming Notes

Software must ensure all the write caches are flushed through a stalling PIPE_CONTROL command followed by another PIPE_CONTROL command to invalidate read only caches prior to programming MI_PIPELINE_SELECT command to change the Pipeline Select Mode. Example: ... Workload-3Dmode PIPE_CONTROL (CS Stall, Depth Cache Flush Enable, Render Target Cache Flush Enable, DC Flush Enable) PIPE_CONTROL (Constant Cache Invalidate, Texture Cache Invalidate, Instruction Cache Invalidate, State Cache invalidate) PIPELINE_SELECT (GPGPU)

Software must clear the **COLOR_CALC_STATE Valid** field in 3DSTATE_CC_STATE_POINTERS command prior to send a PIPELINE_SELECT with **Pipeline Select** set to GPGPU.

Render CS Only: SW must always program PIPE_CONTROL with CS Stall and Render Target Cache Flush Enable set prior to programming PIPELINE_SELECT command for GPGPU workloads i.e when pipeline mode is set to GPGPU. This is required to achieve better GPGPU preemption latencies for certain programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be trade off against performance by not implementing this programming note.

Hardware Binding Tables are only supported for 3D workloads. Resource streamer must be enabled only for 3D workloads. Resource streamer must be disabled for Media and GPGPU workloads. Batch buffer containing both 3D and GPGPU workloads must take care of disabling and enabling Resource Streamer appropriately while changing the PIPELINE_SELECT mode from 3D to GPGPU and vice versa. Resource streamer must be disabled using MI_RS_CONTROL command and Hardware Binding Tables must be disabled by programming 3DSTATE_BINDING_TABLE_POOL_ALLOC with "Binding Table Pool Enable" set to disable (i.e. value '0'). Example below shows disabling and enabling of resource streamer in a batch buffer for 3D and GPGPU workloads. MI_BATCH_BUFFER_START (Resource Streamer Enabled) PIPELINE_SELECT (3D) 3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled) 3D WORKLAOD MI_RS_CONTROL (Disable Resource Streamer) 3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Disabled) PIPELINE_SELECT (GPGPU) GPGPU Workload 3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled) MI_RS_CONTROL (Enable Resource Streamer) 3D WORKLOAD MI_BATCH_BUFFER_END

Render command streamer (RCS) does Fixed Function DOP CG for media and gpgpu workloads. During context restore 3D semi pipeline state is restored to WM, which triggers implicit flush. Theoretically under certain conditions RCS can trigger FFDOP CG while WM in doing implicit flush leading to hangs. Only way to work around this issue is by disabling FFDOP CG feature. This issue is being fixed on G0 stepping.

PIPELINE_SELECT

So far this issue is not yet hit in pre-silicon validation, emulation, Silicon-SV or Silicon-Driver being a rare condition. Work around to disable FFDOP CG is not being applied at this point to allow valdiaiton to make progress on FFDOP CG feature, further decision will be taken when this issue occurs. Validation teams are requested to triage any GPGPU workload hangs with WM no done with FFDOP CG disabled to eliminate this scenario.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE Format: OpCode
	28:27	**Command SubType** Default Value: 1h GFXPIPE_SINGLE_DW Format: OpCode
	26:24	**3D Command Opcode** Format: OpCode Value 1h: GFXPIPE_NONPIPELINED **[Default]** — Project: CHV, BSW
	23:16	**3D Command Sub Opcode** Default Value: 04h GFXPIPE Format: OpCode
	6	**Reserved** Project: CHV, BSW
	15:2	**Reserved** Project: CHV, BSW
	1:0	**Pipeline Selection** 0 — 3D — 3D pipeline is selected 1 — Media — Media pipeline is selected (Includes HD optical disc playback, HD video playback, and generic media workloads) 2 — GPGPU — GPGPU pipeline is selected

Plane

pln - Plane

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The pln instruction computes a component-wise plane equation (w = p*u+q*v+r where u/v/w are vectors and p/q/r are scalars) of src0 and src1 and stores the results in dst. src1 is the input vector u. src0 provides input scalars p, q, and r, where p is the scalar value based on the region description of src0 and q and r are the scalar values implied from the src0 region. Specifically, q is the second component and r is the fourth component of the 4-tuple (128-bit aligned) that p belongs to.

Format: [(pred)] pln[.cmod] (exec_size) dst src0 src1

Restriction	Project
This is a specialized instruction that only supports an execution size (ExecSize) of 8 or 16.	
The src0 region must be a replicated scalar (with HorzStride == VertStride == 0).	
src0 must specify .0 or .4 as the subregister number, corresponding to a subregister byte offset of 0 or 16.	
Source operands cannot be accumulators.	CHV, BSW

Syntax

[(pred)] pln[.cmod] (exec_size) reg reg reg

Pseudocode

```
 Evaluate(WrEn);
for ( n = 0; n < exec_size; n++ ) {
    float dwP = src0.RegNum.SubRegNum[bits4:2];          // A DWord-aligned scalar.
    float dwQ = src0.RegNum.(SubRegNum[bit4:2] | 0x1);   // Second component.
    float dwR = src0.RegNum.(SubRegNum[bit4:2] | 0x3);   // Fourth component.
    if ( ExecSize == 8 ) {
        u = src1.RegNum
        v = src1.(RegNum + 1)
    } else {
        if ( n < 8 ) {
            u = src1.RegNum
            v = src1.(RegNum + 1)
        } else {
            u = src1.(RegNum + 2)
            v = src1.(RegNum + 3)
        }
    }

    if ( WrEn.chan[n] ) {
       dst.chan[n] = dwP * u.chan[n] + dwQ * v.chan[n] + dwR;
    }
}
```

pln - Plane

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	N

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

REP16 Render Target Write MSD

MSD_RTW_REP16 - REP16 Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved** Project: All Format: MBZ Ignored
	30	**Message Precision Subtype** Default Value: 0h Project: All Format: Opcode Full precision data message
	29	**Reserved** Project: All Format: MBZ Ignored
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.

MSD_RTW_REP16 - REP16 Render Target Write MSD

18	**Reserved**
17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message
13	**Reserved**
12	**Last Render Target Select** Project: All Format: Enable This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero. **Programming Notes** When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.
11	**Slot Group Select** Project: All Format: MDC_RT_SGS [CHV, BSW] This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.
10:8	**Render Target Message Subtype** Default Value: 1h Project: All Format: Opcode SIMD16 Single source message with replicated data. Use slots [15:0] for pixel enables, X/Y addresses, and oMask. **Programming Notes** The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].
7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Return

ret - Return

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description

Return execution to the code sequence that called a subroutine. The ret instruction can be predicated or non-predicated. If non-predicated, all channels jump to the return IP in the first channel of src0 and restore CallMask from the second channel of src0. If predicated, the enabled channels jump to the return IP from the first channel of src0 and the corresponding bits in the CallMask are cleared to zero; if all CallMask bits are zero after the ret instruction, then execution jumps to the return IP from the first channel of src0. When SPF is on, the predication control must be scalar.

Format: [(pred)] ret (exec_size) null src0

Restriction

This instruction cannot take accumulator as source.

The src0 regioning control must be <2;2,1>

Syntax

[(pred)] ret (exec_size) null reg

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { PcIP[n] = src0.chan[0]; CallMask[n] = 0; } else { PcIP[n] = IP + 1; } } for ( n = exec_size; n < 32; n++ ) { PcIP[n] = IP + 1; } if ( CallMask[n:0] == 0) ) { // all channels are zero Jump(src0.chan[0]); CallMask = src0.chan[1]; }
```

Predication	Conditional Modifier	Saturation	Source Modifier	Src Types
Y	N	N	N	D, UD

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Round Down

rndd - Round Down

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The rndd instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of src0 and storing the rounded integral float results in dst. This is commonly referred to as the floor() function. Each result follows the rules in the following tables based on the floating-point mode.

Format: [(pred)] rndd[.cmod] (exec_size) dst src0

Syntax

[(pred)] rndd[.cmod] (exec_size) reg reg [(pred)] rndd[.cmod] (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = floor(src0.chan[n]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Round to Nearest or Even

rnde - Round to Nearest or Even

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The rnde instruction takes component-wise floating point round-to-even operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-even increments stored in the rounding increment bits. The round-to-even increment must be added to the results in dst to create the final round-to-even values to emulate the round-to-even operation, commonly known as the round() function. The final results are the one of the two integral float values that is nearer to the input values. If the neither possibility is nearer, the even alternative is chosen. Each result follows the rules in the following tables based on the floating-point mode.

Format: [(pred)] rnde[.cmod] (exec_size) dst src0

Syntax

[(pred)] rnde[.cmod] (exec_size) reg reg [(pred)] rnde[.cmod] (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( src0.chan[n] - floor(src0.chan[n]) > 0.5f ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else if ( src0.chan[n] - floor(src0.chan[n]) < 0.5f ) { dst.chan[n] = floor(src0.chan[n]); } else { if ( floor(src0.chan[n]) is odd ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = floor(src0.chan[n]); } } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Round to Zero

rndz - Round to Zero

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The rndz instruction takes component-wise floating point round-to-zero operation of src0 with results in two pieces - a downward rounded integral float results stored in dst and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in dst to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the truncate() function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Format: [(pred)] rndz[.cmod] (exec_size) dst src0

Syntax

[(pred)] rndz[.cmod] (exec_size) reg reg [(pred)] rndz[.cmod] (exec_size) reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = floor(src0.chan[n]); if ( abs(src0.chan[n]) < abs(dst.chan[n]) ) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = floor(src0.chan[n]); } } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Round Up

rndu - Round Up

Project: CHV, BSW

Source: Eulsa

Length Bias: 4

The rndu instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of src0, commonly known as the ceiling() function. Each result follows the rules in the following tables based on the floating-point mode.

Format: [(pred)] rndu[.cmod] (exec_size) dst src0

Syntax

[(pred)] rndu[.cmod] (exec_size) reg reg [(pred)] rndu[.cmod] (exec_size) reg imm32

Pseudocode

Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { if (src0.chan[n] - floor(src0.chan[n]) > 0.0f) { dst.chan[n] = floor(src0.chan[n]) + 1; } else { dst.chan[n] = src0.chan[n]; } } }

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
F	F

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Scattered Move

smov - Scattered Move

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The smov instruction moves the components in src0 into dst. For each enabled channel, copy src0 to dst. The immediate is used to selectively enable channels without using flags. When predication is enabled, the predicate mask is not generated from the flags. Instead, the immediate is used to mask the execution mask. If any channel is enabled as a result of this masking, the instruction is executed. When predication is not enabled, the immediate masks the execution mask. This provides flexibility to mask out any channel with an immediate.

Format: [(pred)] smov[.cmod] (exec_size) dst src0 src1

Programming Notes

When predication is disabled, the immediate provides the flexibility to perform a select operation without the use of flags.

When predication is enabled, the usage model provides flexibility to select any bit in the flag registers for predication for execution size of 1.

Syntax

[(pred)] smov[.cmod] (exec_size) reg reg imm32

Pseudocode

```
if pred emask = OR (emask AND imm32) Else pmask = imm32. Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if ( WrEn.chan[n] ) { dst.chan[n] = src0.chan[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Scratch Block Read MSD

MSD0R_HWB - Scratch Block Read MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Project: All Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 0h Project: All Format: Opcode Scratch Block Read message
	16	**Channel Mode** Project: CHV, BSW Format: MDC_CMODE [CHV, BSW] Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	15	**Invalidate After Read** Project: All Format: MDC_IAR [CHV, BSW] Specifies if L3 cache lines accessed by the message should be invalidated after the read occurs
	14	**Reserved**

MSD0R_HWB - Scratch Block Read MSD

	13:12	**Data Elements** Project: All Format: MDC_DB_HW [CHV, BSW] Specifies the number of registers to be read or written
	11:0	**Address Offset** Project: All Format: GeneralStateOffset[17:6] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Scratch Block Write MSD

MSD0W_HWB - Scratch Block Write MSD

Project: CHV, BSW
Source: DataPort 0
Length Bias: 1
Family: Block R/W
Group: HW Block R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Scratch Block Message** Default Value: 1h Project: All Format: Opcode Scratch Block Message
	17	**Operation Type** Default Value: 1h Project: All Format: Opcode Scratch Block Write message
	16	**Channel Mode** Project: CHV, BSW Format: MDC_CMODE [CHV, BSW] Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	15:14	**Reserved**
	13:12	**Data Elements** Project: All Format: MDC_DB_HW [CHV, BSW] Specifies the number of registers to be read or written
	11:0	**Address Offset** Project: All Format: GeneralStateOffset[17:6] HWORD (32 byte) based address offset to the BufferAddress in the Message Header.

Select

sel - Select

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

Description

The sel instruction selectively moves the components in src0 or src1 into the channels of dst based on the predication. On a channel by channel basis, if the channel condition is true, data in src0 is moved into dst. Otherwise, data in src1 is moved into dst.

As the predication is used to select the two sources, it is not included in the evaluation of WrEn. The predicate clause is mandatory if cmod is omitted/0000b. If both predication and the conditional modifier are omitted, the results are undefined.

If the conditional modifier is specified (not 0000b, a compare is performed and the resulting condition flag is used for the sel instruction. Conditional modifiers .ge and .l follow the cmpn rules, and all other conditional modifiers follow the cmp rules. Predication is not allowed in this mode.

A sel instruction with cmod .l is used to emulate a MIN instruction.

A sel instruction with cmod .ge is used to emulate a MAX instruction.

For a sel instruction with a .l or .ge conditional modifier, if one source is NaN and the other not NaN, the non-NaN source is the result. If both sources are NaNs, the result is NaN. For all other conditional modifiers, if either source is NaN then src1 is selected.

A sel instruction without a conditional modifier always copies a denorm source value to a denorm destination value (in the manner of a raw move). This applies even if the source modifies are set on the *sel* instruction sources.

The sel instruction uses any conditional modifier internally and does not update the flag register if a conditional modifier is used.

A sel instruction with cmod or source modifier will flush denorm to zero, depending on the denorm mode bit; a sel instruction without cmod and source modifier will retain denorm.

Format: (pred) sel[.cmod] (exec_size) dst src0 src1

Restriction

Predicated sel instruction cannot be used in mixed mode operation with packed half float destination.

Syntax

(pred) sel[.cmod] (exec_size) reg reg reg (pred) sel[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn, NoPMask); if (cmod == "0000") { // no CMod Evaluate(PMask); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( PMask.channel[n] ) { dst.chan[n] = src0.chan[n]; } else { dst.chan[n] = src1.chan[n]; } } } } else { // with CMod Evaluate(CMod); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { if ( CMod.chan[n] ) { dst.chan[n] = src0.chan[n]; } else { dst.chan[n] = src1.chan[n]; } } } }
```

sel - Select

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
*B,*W*D	*B,*W,*D	
F	F	
DF	DF	CHV, BSW
*W,*D,*Q	*W,*D,*Q	CHV, BSW
HF	HF	CHV, BSW
HF, F	HF, F	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Shift Left

shl - Shift Left

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

Description

Perform component-wise logical left shift of the bits in src0 by the shift count indicated in src1, storing the results in dst, inserting zero bits in the number of LSBs indicated by the shift count. Hardware detects overflow properly and uses it to perform any saturation operation on the result, as long as the shifted result is within 33 bits. Otherwise, the result is undefined. Note: For word and DWord operands, the accumulators have 33 bits.

In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.

Format: [(pred)] shl[.cmod] (exec_size) dst src0 src1

Restriction

Accumulator cannot be destination, implicit or explicit.

Syntax

[(pred)] shl[.cmod] (exec_size) reg reg reg [(pred)] shl[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n++ ) { if ( WrEn.chan[n] ) { shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F dst.chan[n] = src0.chan[n] « shiftCnt; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier	Src Types	Dst Types
Y	Y	Y	Y	*B,*W,*D	*B,*W,*D

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Shift Right

shr - Shift Right

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

Description
Perform component-wise logical right shift with zero insertion of the bits in src0 by the shift count indicated in src1, storing the results in dst. Insert zero bits in the number of MSBs indicated by the shift count. src0 and dst can have different types and can be signed or unsigned. Note: For word and DWord operands, the accumulators have 33 bits. Note: For unsigned src0 types, shr and asr produce the same result.
In QWord mode, the shift count is taken from the low six bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 63. Otherwise the shift count is taken from the low five bits of src1 regardless of the src1 type and treated as an unsigned integer in the range 0 to 31. The operation uses QWord mode if src0 or dst has the Q or UQ type but not if src1 is the only operand with the Q or UQ type.
Format: [(pred)] shr[.cmod] (exec_size) dst src0 src1

Syntax
[(pred)] shr[.cmod] (exec_size) reg reg reg [(pred)] shr[.cmod] (exec_size) reg reg imm32

Pseudocode	Project
Evaluate(WrEn); for (n = 0; n < exec_size; n++) { if (WrEn.chan[n]) { shiftCnt = src0 or dst has Q or UQ type ? src1.chan[n] & 0x3F : src1.chan[n] & 0x1F dst.chan[n] = src0.chan[n] » shiftCnt; } }	CHV, BSW

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types	Project
UB, UW, UD	UB, UW, UD	
UW, UD, UQ	UW, UD, UQ	CHV, BSW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

SIMD8 Render Target Write MSD

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

Project:	CHV, BSW
Source:	Render Cache DataPort
Length Bias:	1
Family:	Other
Group:	Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 0h Project: All Format: Opcode Full precision data message
	29	**Reserved**
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message

MSD_RTW_SIMD8 - SIMD8 Render Target Write MSD

13 **Reserved**

Project:	CHV, BSW
Format:	MBZ

Ignored

12 **Last Render Target Select**

Project:	All
Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 **Slot Group Select**

Project:	All
Format:	MDC_RT_SGS [CHV, BSW]

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

10:8 **Render Target Message Subtype**

Default Value:	4h
Project:	All
Format:	Opcode

SIMD8 single source message. Use slots [7:0] for pixel enables, X/Y addresses, and oMask.

Programming Notes

The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [23:16] are referenced instead of [7:0].

7:0 **Binding Table Index**

Project:	All
Format:	MDC_BTS [CHV, BSW]

Specifies the Binding Table Index for the message

SIMD16 Render Target Write MSD

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

Project: CHV, BSW
Source: Render Cache DataPort
Length Bias: 1
Family: Other
Group: Render Target R/W

DWord	Bit	Description
0	31	**Reserved**
	30	**Message Precision Subtype** Default Value: 0h Project: All Format: Opcode Full precision data message
	29	**Reserved**
	28:25	**Message Length** Project: All Format: U4 Specifies the number of 256-bit GRF registers sent as the message payload (including the header). Valid value ranges are 1 to 15.
	24:20	**Response Length** Project: All Format: U5 Specifies the number of 256-bit GRF registers expected as the message response payload. Valid value ranges are 0 to 16.
	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the 2-register header.
	18	**Reserved**
	17:14	**Message Type** Default Value: 0Ch Project: All Format: Opcode Render Target Write message

MSD_RTW_SIMD16 - SIMD16 Render Target Write MSD

13 **Reserved**

Project:	CHV, BSW
Format:	MBZ

Ignored

12 **Last Render Target Select**

Project:	All
Format:	Enable

This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. In general, when threads are not launched by 3D FF, this bit must be zero.

Programming Notes

When a pixel shader has render target writes at finer granularity than the dispatch rate, last render target write to a null surface must be present at the dispatch rate with this bit set. In particular, if a kernel is dispatched at pixel rate and it only writes to render targets at sample-rate, it must include a pixel-rate render target write to a null surface with Last Render Target Select bit enabled.

11 **Slot Group Select**

Project:	All
Format:	MDC_RT_SGS [CHV, BSW]

This field selects whether slots 15:0 or slots 31:16 are used for bypassed data.

10:8 **Render Target Message Subtype**

Default Value:	0h
Project:	All
Format:	Opcode

SIMD16 Single source message. Use slots [15:0] for pixel enables, X/Y addresses, and oMask.

Programming Notes

The above slots indicated are within the 16 slots selected by Slot Group Select. If SLOTGRP_HI is selected, slots [31:16] are referenced instead of [15:0].

7:0 **Binding Table Index**

Project:	All
Format:	MDC_BTS [CHV, BSW]

Specifies the Binding Table Index for the message

STATE_BASE_ADDRESS

STATE_BASE_ADDRESS

Project: CHV, BSW
Source: PRM
Length Bias: 2

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE.
For more information see the Base Address Utilization table in the Memory Access Indirection narrative topic.

Programming Notes	Project
The following commands must be reissued following any change to the base addresses: • 3DSTATE_CC_POINTERS • 3DSTATE_BINDING_TABLE_POINTERS • 3DSTATE_SAMPLER_STATE_POINTERS • 3DSTATE_VIEWPORT_STATE_POINTERS • MEDIA_STATE_POINTERS Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.	
SW must always program PIPE_CONTROL with "CS Stall" and "Render Target Cache Flush Enable" set before programming STATE_BASE_ADDRESS command for GPGPU workloads i.e when pipeline select is GPGPU via PIPELINE_SELECT command. This is required to achieve better GPGPU preemption latencies in certain workload programming sequences. If programming PIPE_CONTROL has performance implications then preemption latencies can be traded off against performance by not implementing this programming note.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 01h STATE_BASE_ADDRESS
	15:8	**Reserved** Project: All Format: MBZ

STATE_BASE_ADDRESS

7:0 DWord Length

Project:	CHV, BSW
Format:	=n Total Length - 2

Value	Name	Description
Eh	DWORD_COUNT_n **[Default]**	Excludes DWord (0,1)

1..2 63:12 General State Base Address

Project:	All
Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for general state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

Programming Notes

Bounds checking is performed on general state accesses by Data Port Shared Functions for stateless A32 messages.
Bounds checking is enabled when General State Base Address [46:12] + General State Buffer Size [31:12] is <= 2^47. This ensures that the General State Buffer does not straddle the canonical address boundary where GraphicsAddress [47] changes.

Restriction

General State Base Address [47:12] + General State Buffer Size [31:12] must be < 2^48. It is illegal programming for this to be >= 2^48.

When using stateless (A32) Data Port messages, General State Base Address [47:12] + Buffer Base Address [31:0] must be < 2^48. It is illegal for this to be >= 2^48.

11 Reserved

Project:	All
Format:	MBZ

10:4 General State Memory Object Control State

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect state using the **General State Base Address**, with the exception of the stateless data port accesses.

3:1 Reserved

0 General State Base Address Modify Enable

Project:	All
Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated address.	All
1h	Enable	Modify the address.	All

STATE_BASE_ADDRESS

3 — 31:23 — Reserved

Project:	All
Format:	MBZ

22:16 — Stateless Data Port Access Memory Object Control State

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for stateless data port accesses.

15:0 — Reserved

Project:	All
Format:	MBZ

4..5 — 63:12 — Surface State Base Address

Project:	All
Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for binding table and surface state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

11 — Reserved

Project:	All
Format:	MBZ

10:4 — Surface State Memory Object Control State

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect state using the **Surface State Base Address**.

3:1 — Reserved

Project:	All
Format:	MBZ

0 — Surface State Base Address Modify Enable

Project:	All
Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated address.	All
1h	Enable	Modify the address.	All

STATE_BASE_ADDRESS

		Programming Notes Setting this bit to 1 in a batch buffer causes the resource streamer to stop; for performance reasons the SW should only place commands with this bit set in the ring buffer. Before programming the Surface State Base Address, the RS must be disabled. Within a batch buffer where the RS is enabled, RS may be disabled thru a MI_RS_CONTROL command with Resource Streamer Control cleared prior to the STATE_BASE_ADDRESS with Surface State Base Address Modify Enable set and then re-enabled with another MI_RS_CONTROL with Resource Streamer Control set.
6..7	63:12	**Dynamic State Base Address** Project: All Format: GraphicsAddress[63:12] Specifies the 4K-byte aligned base address for sampler and viewport state accesses. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].
	11	**Reserved** Project: All Format: MBZ
	10:4	**Dynamic State Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for indirect state using the **Dynamic State Base Address**. Push constants defined in 3DSTATE_CONSTANT_(VS | GS | PS) commands do not use this control state, although they can use the corresponding base address. The memory object control state for push constants is defined within the command.
	3:1	**Reserved** Project: All Format: MBZ
	0	**Dynamic State Base Address Modify Enable** Project: All Format: Enable The other fields in this DWord and the following DWord are updated only when this bit is set. **Value** / **Name** / **Description** / **Project** 0h / Disable / Ignore the updated address. / All 1h / Enable / Modify the address. / All
8..9	63:12	**Indirect Object Base Address** Project: All Format: GraphicsAddress[63:12]IndirectObject Specifies the 4K-byte aligned base address for indirect object load in MEDIA_OBJECT command.

STATE_BASE_ADDRESS

11 — Reserved

Project:	All
Format:	MBZ

10:4 — Indirect Object Memory Object Control State

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for indirect objects using the **Indirect Object Base Address**.

3:1 — Reserved

Project:	All
Format:	MBZ

0 — Indirect Object Base Address Modify Enable

Project:	All
Format:	Enable

The other fields in this DWord and the following DWord are updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated address.	All
1h	Enable	Modify the address.	All

10..11 — 63:12 — Instruction Base Address

Project:	All
Format:	GraphicsAddress[63:12]

Specifies the 4K-byte aligned base address for all EU instruction accesses.
GraphicsAddress[63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47].

11 — Reserved

Project:	All
Format:	MBZ

10:4 — Instruction Memory Object Control State

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for EU instructions using the **Instruction Base Address**.

3:1 — Reserved

Project:	All
Format:	MBZ

STATE_BASE_ADDRESS

	0	**Instruction Base Address Modify Enable** Project: All Format: Enable The other fields in this DWord and the following DWord are updated only when this bit is set. 0h – Disable – Ignore the updated address. – All 1h – Enable – Modify the address. – All
12	31:12	**General State Buffer Size** Project: All Format: U20 FormatDesc This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer. Workaround – Project When the General State Buffer Size is programmed to 0, SLM accesses are treated as out-of-bounds (should only apply to Stateless accesses). Workaround is to program the General State Buffer Size to a value > 0. – CHV, BSW If Per Thread Scratch Space Size bounds checking is enabled by GT_MODE[15], then General State Buffer Size must be set larger than the maximum Per Thread Scratch Space Size. – CHV, BSW [CHV, BSW]
	11:1	**Reserved**
	0	**General State Buffer Size Modify Enable** Project: All Format: Enable The bound in this DWord is updated only when this bit is set. 0h – Disable – Ignore the updated bound. – All 1h – Enable – Modify the updated bound. – All
13	31:12	**Dynamic State Buffer Size** Project: All Format: U20 FormatDesc This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.

STATE_BASE_ADDRESS

	11:1	**Reserved**
	0	**Dynamic State Buffer Size Modify Enable** Project: All Format: Enable FormatDesc: The bound in this DWord is updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated bound.	All
1h	Enable	Modify the updated bound.	All

14	31:12	**Indirect Object Buffer Size** Project: All Format: U20 FormatDesc: This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.
	11:1	**Reserved**
	0	**Indirect Object Buffer Size Modify Enable** Project: All Format: Enable FormatDesc: The bound in this DWord is updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated bound.	All
1h	Enable	Modify the updated bound.	All

15	31:12	**Instruction Buffer Size** Project: All Format: U20 FormatDesc: This field specifies the size of the buffer in 4K pages. Any access that straddles or goes past the end of the buffer returns 0. Note that BufferSize=0 indicates that there is no valid data in the buffer.
	11:1	**Reserved**

STATE_BASE_ADDRESS

	0	**Instruction Buffer size Modify Enable** Project: All Format: Enable FormatDesc: The bound in this DWord is updated only when this bit is set.

Value	Name	Description	Project
0h	Disable	Ignore the updated bound.	All

STATE_PREFETCH

STATE_PREFETCH

Project: CHV, BSW

Source: PRM

Length Bias: 2

(This command is provided strictly for performance optimization opportunities, and likely requires some experimentation to evaluate the overall impact of additional prefetching.)

The STATE_PREFETCH command causes the GPE to attempt to prefetch a sequence of 64-byte cache lines into the GPE-internal cache ("L2 ISC") used to access EU kernel instructions and fixed/shared function indirect state data. While state descriptors, surface state, and sampler state are automatically prefetched by the GPE, this command may be used to prefetch data not automatically prefetched, such as: 3D viewport state; Media pipeline Interface Descriptors; EU kernel instructions.

Restriction	Project
Restriction: Due to know HW issue this command doesn't achieve its intended purpose and must not be exercised/programmed by SW.	CHV, BSW

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 0h GFXPIPE_PIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 03h STATE_PREFETCH
	15:8	**Reserved**
	7:0	**DWord Length** Project: All Format: =n Total Length - 2 Value: 0h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31:6	**Prefetch Pointer** Project: All Format: GraphicsAddress[31:6] Specifies the 64-byte aligned address to start the prefetch from. This pointer is an absolute virtual address, it is not relative to any base pointer.
	5:3	**Reserved**

STATE_PREFETCH

	2:0	**Prefetch Count** Project: All Format: U3-1 count of cache lines Indicates the number of contiguous 64-byte cache lines that will be prefetched.

Value	Name	Description
[0,7]		indicating a count of [1,8]

STATE_SIP

STATE_SIP

Project: All
Source: PRM
Length Bias: 2

The STATE_SIP command specifies the starting instruction location of the System Routine that is shared by all threads in execution.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 02h STATE_SIP
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**DWord Length** Project: All Format: =n Total Length - 2 Value: 1h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1) — Project: CHV, BSW
1..2 **Project:** CHV, BSW	63:4	**System Instruction Pointer** Project: All Format: InstructionBaseOffset[63:4]Kernel Specifies the instruction address of the system routine associated with the current context as a 128-bit granular offset from the Instruction Base Address. SIP is shared by all threads in execution. The address specifies the double quadword aligned instruction location. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** This portion of the command is not context save/restored. The context image may restore this command as a 2 dword command rather than a 3 dword command.
	3:0	**Reserved**

Sum of Absolute Difference 2

sad2 - Sum of Absolute Difference 2

Project: CHV, BSW

Source: EuIsa

Length Bias: 4

The sad2 instruction takes source data channels from src0 and src1 in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between src0 and src1 and stores the scalar result in the first channel of the 2-tuple in dst. The results are also stored in the accumulator register. The destination operand and the accumulator maintain 16 bits per channel precision. The destination register must be aligned to even word (DWord). The even words in the destination region will contain the correct data. The odd words are also written but with undefined values.

Format: [(pred)] sad2[.cmod] (exec_size) dst src0 src1

Restriction

Source operands cannot be accumulators.

The execution size cannot be 1 as the computation requires at least two data channels.

Syntax

[(pred)] sad2[.cmod] (exec_size) reg reg reg [(pred)] sad2[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 2 ) { if ( WrEn.chan[n] ) { dst.chan[n] = abs(src0.chan[n] - src1.chan[n]) + abs(src0.chan[n+1] - src1.chan[n+1]); } }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	Y	Y	Y

Src Types	Dst Types
B, UB	W, UW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

Sum of Absolute Difference Accumulate 2

sada2 - Sum of Absolute Difference Accumulate 2

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

The sada2 instruction takes source data channels from src0 and src1 in groups of 2-tuples. For each 2-tuple, it computes the sum-of-absolute-difference (SAD) between src0 and src1, adds the intermediate result with the accumulator value corresponding to the first channel, and stores the scalar result in the first channel of the 2-tuple in dst. The destination operand and the accumulator maintain 16 bits per channel precision. Higher precision (guide bits) stored in the accumulator allows up to 64 rounds of sada2 instructions to be issued back to back without overflowing the accumulator. The destination register must be aligned to even word (DWord). The even words in the destination region will contain the correct data. The odd words are also written but with undefined values.

Format: [(pred)] sada2[.cmod] (exec_size) dst src0 src1

Restriction

Source operands cannot be accumulators.

The execution size cannot be 1 as the computation requires at least two data channels.

Syntax

[(pred)] sada2[.cmod] (exec_size) reg reg reg [(pred)] sada2[.cmod] (exec_size) reg reg imm32

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < exec_size; n += 2 ) { uwTmp = abs(src0.chan[n] - src1.chan[n]) +
abs(src0.chan[n+1] - src1.chan[n+1]); if ( WrEn.chan[n] ) { dst.chan[n] = uwTmp + acc[n]; } }
```

Predication	Conditional Modifier	Saturation	Source Modifier	Src Types	Dst Types
Y	Y	Y	Y	B, UB	W, UW

DWord	Bit	Description
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

SWTESS_BASE_ADDRESS

SWTESS_BASE_ADDRESS

Project: CHV, BSW
Source: PRM
Length Bias: 2

The SWTESS_BASE_ADDRESS command sets the base pointers for SW Tessellation data read access by the TE unit.

Programming Notes

This base address must also be comprehended in the SURFACE_STATE used by the HS kernel to write the SW tessellation data.
Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h GFXPIPE
	28:27	**Command SubType** Default Value: 0h GFXPIPE_COMMON
	26:24	**3D Command Opcode** Default Value: 1h GFXPIPE_NONPIPELINED
	23:16	**3D Command Sub Opcode** Default Value: 03h SWTESS_BASE_ADDRESS
	15:8	**Reserved**
	7:0	**DWord Length** Project: All Format: =n Total Length - 2 Value: 0h — Name: DWORD_COUNT_n **[Default]** — Description: Excludes DWord (0,1)
1	31:12	**SW Tessellation Base Address** Project: All Format: GraphicsAddress[31:12] Specifies the 4K-byte aligned base address for TE unit SW tessellation data read accesses.
	11:8	**SW Tessellation Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state used by the TE unit to read SW tessellation data from memory.

SWTESS_BASE_ADDRESS		
	7:0	**Reserved** Project: All Format: MBZ
2 **Project:** CHV, BSW	31:16	**Reserved** Project: All Format: MBZ
	15:0	**SW Tessellation Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] Specifies most significant bits of the 4K-byte aligned base address for TE unit SW tessellation data read accesses. See SW Tessellation Base Address[31:12] in DWord 0.

Typed Surface Read MSD

MSD1R_TS - Typed Surface Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Surface R/W
Group:	Scattered Typed Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 05h Project: All Format: Opcode Typed Surface Read message
	13:12	**Slot Group** Project: All Format: MDC_SG3 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Project: All Format: MDC_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Typed Surface Write MSD

MSD1W_TS - Typed Surface Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Typed Surface R/W
Group:	Scattered Typed Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 0Dh Project: All Format: Opcode Typed Surface Write message
	13:12	**Slot Group** Project: All Format: MDC_SG3 [CHV, BSW] Specifies the Slot Group mode of the message (which slots are processed)
	11:8	**Channel Mask** Project: All Format: MDC_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS [CHV, BSW] Specifies the Binding Table Index for the message

Untyped Surface Read MSD

MSD1R_US - Untyped Surface Read MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 01h Project: All Format: Opcode Untyped Surface Read message
	13:12	**SIMD Mode** Project: All Format: MDC_SM3 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Project: All Format: MDC_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

Untyped Surface Write MSD

MSD1W_US - Untyped Surface Write MSD

Project:	CHV, BSW
Source:	DataPort 1
Length Bias:	1
Family:	Untyped Surface R/W
Group:	Scattered Untyped Surface R/W

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHP [CHV, BSW] If set, indicates that the message includes the header.
	18:14	**Message Type** Default Value: 09h Project: All Format: Opcode Untyped Surface Write message
	13:12	**SIMD Mode** Project: All Format: MDC_SM3 [CHV, BSW] Specifies the SIMD mode of the message (number of slots processed)
	11:8	**Channel Mask** Project: All Format: MDC_UW_CMASK [CHV, BSW] Specifies which RGBA channels are included in the message payload.
	7:0	**Binding Table Index** Project: All Format: MDC_BTS_SLM_A32 [CHV, BSW] Specifies the Binding Table Index for the message

URB Hword Dual Block Read MSD

MSD_UR_HWDB - URB Hword Dual Block Read MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved**
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved**
	15	**Swizzle Control** Project: All Format: Opcode Value 1 – URB_INTERLEAVED **[Default]** – Use two URB entries (**URB Handle 0** and **URB Handle 1**). – Project: All
	14:4	**Global Offset** Project: All Format: U11 Specifes the offset, in units of Hword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,1023]
	3:0	**URB Opcode** Project: All Format: Opcode Value 2 – URB_READ_HWORD **[Default]** – URB Hword Read message – Project: All

URB Hword Dual Block Write MSD

MSD_UW_HWDB - URB Hword Dual Block Write MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved**
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved**
	15	**Swizzle Control** Project: All Format: Opcode **Value** 1 — **Name** URB_INTERLEAVED **[Default]** — **Description** Use two URB entries (**URB Handle 0** and **URB Handle 1**). — **Project** All
	14:4	**Global Offset** Project: All Format: U11 Specifes the offset, in units of Hword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,1023]
	3:0	**URB Opcode** Project: All Format: Opcode **Value** 0 — **Name** URB_WRITE_HWORD **[Default]** — **Description** URB Hword Read message — **Project** All

URB Oword Block Write MSD

MSD_UW_OWB - URB Oword Block Write MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved**
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved**
	15	**Swizzle Control** Project: All Format: Opcode Value 0 — URB_NOSWIZZLE **[Default]** — Use a single URB entry (**URB Handle 0**). — All
	14:4	**Global Offset** Project: All Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Project: All Format: Opcode Value 1 — URB_WRITE_OWORD **[Default]** — URB Oword Write message — All

URB Oword Dual Block Read MSD

MSD_UR_OWDB - URB Oword Dual Block Read MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved**
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved**
	15	**Swizzle Control** Project: All Format: Opcode **Value** 1 — **Name** URB_INTERLEAVED **[Default]** — **Description** Use two URB entries (**URB Handle 0** and **URB Handle 1**). — **Project** All
	14:4	**Global Offset** Project: All Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Project: All Format: Opcode **Value** 3 — **Name** URB_READ_OWORD **[Default]** — **Description** URB Oword Read message — **Project** All

URB Oword Dual Block Write MSD

MSD_UW_OWDB - URB Oword Dual Block Write MSD

Project: CHV, BSW
Source: Read-Only DataPort
Length Bias: 1

DWord	Bit	Description
0	19	**Header Present** Project: All Format: MDC_MHR [CHV, BSW] Indicates that the message requires a header.
	18	**Reserved**
	17	**Per Slot Offset** Format: MHC_PSOP Specifies if per-slot offsets are present and will be added to the **Global Offset**.
	16	**Reserved**
	15	**Swizzle Control** Project: All Format: Opcode Value 1 – URB_INTERLEAVED **[Default]** – Use two URB entries (**URB Handle 0** and **URB Handle 1**). – Project: All
	14:4	**Global Offset** Project: All Format: U11 Specifes the offset, in units of Oword elements, from the start of the URB handle for the access. If **Per Slot Offset** is set, the global offset is added to those offsets to form the overall offset. Range [0,2047]
	3:0	**URB Opcode** Project: All Format: Opcode Value 1 – URB_WRITE_OWORD **[Default]** – URB Oword Write message – Project: All

VEBOX_STATE

VEBOX_STATE

Project: CHV, BSW
Source: VideoEnhancementCS
Length Bias: 2

This command controls the internal functions of the VEBOX. This command has a set of indirect state buffers:

- DN/DI state
- Capture Pipe state

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h Format: OpCode
	20:16	**SubOpcode B** Default Value: 2h Format: OpCode
	15:12	**Reserved**
	11:0	**DWord Length** Format: =n Total Length - 2 Value: Ah, Name: , Description: (Excludes DWords 0, 1)
1	31:25	**State Surface Control Bits**
	24:23	**Reserved**
	22	**Reserved**
	21	**Reserved**
	20	**Reserved**

VEBOX_STATE

	19:15	**Reserved**
	14	**Single Slice VEBOX Enable** Project: CHV, BSW For products that have 2 entire VEBOXes that automatically split the frame, this enable emulates a 1 VEBOX product, running at 1/2 speed and only outputting a single set of per command statistics.
	13	**Hot Pixel Filtering Enable** Enables hot pixel detection/filtering.
	12	**Alpha Plane Enable** Enables the reading of an independent Alpha plane. Mutually exclusive with Vignette Enable. If **Alpha from State Select** is set it overrides this bit.
	11	**Reserved**
	10	**Demosaic Enable** The Demosaic will be used, and White balance statistics will be gathered. The Capture Pipe State Table will be read. This bit is mutually exclusive with **DI Enable.**
	9:8	**DI Output Frames** Indicates which frames to output in DI mode.

Value	Name
00b	Output Both Frames
01b	Output Previous Frame Only
10b	Output Current Frame Only

Programming Notes
Field is ignored if DI Enable = 0. If Previous Frame Only or Current Frame Only are selected, then the **LACE Single Histogram Set** must not try to collect a histogram from the disabled frame.
Field must be programmed to 10 (Output Current Frame Only) for DI First Frame.

	7	**444 -> 422 Downsample Method**

Value	Name
1	Average horizontally aligned chromas
0	Drop right chroma of the pair **[Default]**

Programming Notes

444->422	422->420	Description
0	0	No averaging, only down sampling
0	1	Not Supported
1	0	Only Horizontal averaging
1	1	Horizontal and Vertical averaging

VEBOX_STATE

	6	**422 -> 420 Downsample Method** Project: CHV, BSW Value 1: Average vertically aligned chromas Value 0: Drop lower chroma of the pair **[Default]** **Programming Notes**: To enable averaging in case of 420 (NV12/P016) output formats, 444->422 and 422->420 should be set.
	5	**DN/DI First Frame** Format: Enable Indicates that this is the first frame of the stream, so previous clean is not available. Value 0: Not first field; previous clean surface state is valid Value 1: First field; previous clean surface state is invalid **Programming Notes**: If both DN and DI are disabled, this bit must be 0.
	4	**DI Enable** Format: Enable Deinterlacer is bypassed if this is disabled: the output is the same as the input (same as a 2:2 cadence). FMD and STMM are not calculated and the values in the response message are 0. Value 0: Do not calculate DI Value 1: Calculate DI
	3	**DN Enable** Format: Enable Denoise is bypassed if this is low - BNE is still calculated and output, but the denoised fields are not. VDI does not read in the denoised previous frame but uses the pointer for the original previous frame. Value 0: Do not denoise frame Value 1: Denoise frame **Programming Notes**: If DN and/or Hotpixel are the only functions enabled then the only output is the Denoised Output which is the same surface format as the input.
	2	**Reserved**

VEBOX_STATE

	1	**Color Gamut Compression Enable** Indicates if the Gamut Compression feature is enabled. If set then the Gamut State will be read. VEB_VERTEXTABLE_STATE is only needed if this bit is set.
	0	**Color Gamut Expansion Enable** Indicates if the Gamut Expansion feature is enabled. If set then the Gamut State will be read.
2	31:12	**DN/DI State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the DN/DI State buffer. This points to a buffer containing the 10 Dwords of the DN/DI state.
	11:0	**Reserved** Format: MBZ
3	31:16	**Reserved** Format: MBZ
	15:0	**DN/DI State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the DN/DI State Buffer.
4	31:12	**Reserved**
	11:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:0	**Reserved**
6	31:12	**Gamut State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Gamut State buffer. This points to a buffer containing the 42 Dwords of Gamut Compression / Gamut Expansion state.
	11:0	**Reserved** Format: MBZ
7	31:16	**Reserved** Format: MBZ
	15:0	**Gamut State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Gamut State Buffer.
8	31:12	**Vertex Table State Pointer Low**

VEBOX_STATE

		Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Vertex Table. This points to a buffer containing the 512 Dwords of the Gamut Compression Vertex Table.
	11:0	**Reserved** Format: MBZ
9	31:16	**Reserved** Format: MBZ
	15:0	**Vertex Table State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Vertex State Buffer.
10	31:12	**Capture Pipe State Pointer Low** Format: GraphicAddress[31:12] Bits 31:12 of the starting address of the Capture Pipe State Table. This points to a buffer containing the X Dwords of the Capture Pipe State.
	11:0	**Reserved** Format: MBZ
11	31:16	**Reserved** Format: MBZ
	15:0	**Capture Pipe State Pointer High** Format: GraphicAddress[47:32] Bits 47:32 of the starting address of the Capture Pipe State Table.

VEBOX_SURFACE_STATE

VEBOX_SURFACE_STATE	
Project:	CHV, BSW
Source:	VideoEnhancementCS
Length Bias:	2

Description
The input and output data containers accessed are called "surfaces". Surface state is sent to VEBOX via an inline state command rather than using binding tables. SURFACE_STATE contains the parameters defining each surface to be accessed, including its size, format, and offsets to its subsurfaces. The surface's base address is in the execution command. Despite having multiple input and output surfaces, we limit the number of surface states to one for input surfaces and one for output surfaces. The other surfaces are derived from the input/output surface states.
The Current Frame Input surface uses the Input SURFACE_STATE
The Previous Denoised Input surface uses the Input SURFACE_STATE. (For 12-bit Bayer pattern inputs this will be 8-bit.)
The Current Denoised Output surface uses the Input SURFACE_STATE. (For 12-bit Bayer pattern inputs this will be 8-bit.)
The STMM/Noise History Input surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.
The STMM/Noise History Output surface uses the Input SURFACE_STATE with Tile-Y and Width/Height a multiple of 4.
The FMD per block output / per Frame Output surface uses the Linear SURFACE_STATE (see note below).
The Alpha surface uses the Linear A8 SURFACE_STATE with Width/Height equal to Input Surface. Pitch is width rounded to next 64.
The Vignette Correction surface uses the Linear 16-bit SURFACE_STATE with: Width = 4 * ((Input Width 3)/4) Height = ((Input Height 3)/4) Pitch in bytes is (vignette width*2) rounded to next 64.
The STMM height is the same as the Input Surface height except when the input **Surface Format** is Bayer Pattern and the **Bayer Pattern Offset** is 10 or 11, in which case the height is the input height + 4. For Bayer pattern inputs when the **Bayer Pattern Offset** is 10 or 11, the Current Denoised Output/Previous Denoised Input will also have a height which is the input height + 4. For Bayer pattern inputs only the Current Denoised Output/Previous Denoised Input are in Tile-Y.
The linear surface for FMD statistics is linear (not tiled). The height of the per block statistics is (Input Height +3)/4 - the Input Surface height in pixels is rounded up to the next even 4 and divided by 4. The width of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 16 bytes. The pitch of the per block section in bytes is equal to the width of the Input Surface in pixels rounded up to the next 64 bytes.
The STMM surfaces must be identical to the Input surface except for the tiling mode must be Tile-Y and the pitch must be legal for Tile-Y (increased to the next larger legal pitch). If the input surface is packed (Surface

VEBOX_SURFACE_STATE

Format from 0 to 3 for DN/DI) or 12/10-bit Bayer Pattern then the pitch for the STMM surface is 1/2 the pitch of the input surface (rounded up to the next larger legal Tile-Y pitch). The width and height must be a multiple of 4 rounded up from the input height.

Programming Notes
VEBOX may write to memory between the surface width and the surface pitch for output surfaces.
For 8bit Alpha input, when converting to 16bit output it is padded with 8bit zeros in the LSB.

DWord	Bit	Description
0	31:29	**Command Type** Default Value: 3h PARALLEL_VIDEO_PIPE Format: OpCode
	28:27	**Media Command Pipeline** Default Value: 2h Media Format: OpCode
	26:24	**Media Command OpCode** Default Value: 4h VEBOX Format: OpCode
	23:21	**SubOpcode A** Default Value: 0h VEBOX Format: OpCode
	20:16	**SubOpcode B** Default Value: 0h VEBOX Format: OpCode
	15:12	**Reserved** Format: MBZ
	11:0	**DWord Length** Format: =n Total Length - 2 Value: 4h — Name: DWORD_COUNT_n **[Default]** — Description: (Excludes DWords 0, 1) — Project: CHV, BSW
1	31:1	**Reserved**
	0	**Surface Identification** Specifies which set of surfaces this command refers to: 1 — Output surface (all except the Denoised Current output surface) 0 — Input surface and Denoised Current Output Surface
2	31:18	**Height**

VEBOX_SURFACE_STATE

		Format: U14 This field specifies the height of the surface in units of pixels. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. **Value** / **Name** / **Description** / **Exists If** [15, 16383] / / representing heights [16,16384] / [15, 8191] / / / //Scalar Enabled - For Input surface only [63, 2047] / / / //Scalar + SFC Enabled - For Input surface only **Programming Notes** **Height** (field value + 1) must be a multiple of 2 for PLANAR_420 surfaces. **Height** (field value +1) must be a multiple of 2 when the deinterlace function is enabled (field mode) or when the denoise function is enabled with **Progressive DN** = 0. It must be a multiple of 4 when interleaved deinterlace/denoise and PLANAR_420 are both being used. **VEBOX** supports a minimum height of 16. **Height** (field value + 1) must be a multiple of 2 for Bayer surfaces.
	17:4	**Width** Format: U14 This field specifies the width of the surface in units of pixels. For PLANAR surface formats, this field indicates the width of the Y (luma) plane. **Value** / **Name** / **Description** / **Exists If** [63,16383] / / representing widths [64,16384] / [63,8191] / / / //Scalar Enabled - For Input surface only [63,2047] / / / //Scalar and SFC Enabled - For Input Surface only **Programming Notes** The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the **Surface Pitch** field). **Width** (field value + 1) must be a multiple of 2 for PLANAR_420, PLANAR_422, and all YCRCB_* surfaces, and must be a multiple of 4 for PLANAR_411 surfaces. **VEBOX** supports a minimum width of 64
	3:0	**Reserved**
3	31:28	**Surface Format** Project: CHV, BSW Format: U4 Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1. **Value** / **Name** / **Description**

VEBOX_SURFACE_STATE

4	PLANAR_420_8	NV12 with Interleave Chroma set
14	Bayer pattern	Demosaic input only
15	Reserved	

27 **Interleave Chroma**

Project:	CHV, BSW
Format:	Enable

This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.

26:25 **Bayer Pattern Offset**

Project:	CHV, BSW

Specifies the starting pixel offset for the Bayer pattern used for Capture Pipe.

Value	Name
00b	Pixel at X=0, Y=0 is Blue
01b	Pixel at X=0, Y=0 is Red
10b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Red
11b	Pixel at X=0, Y=0 is Green, Pixel at X=1, Y=0 is Blue

24 **Bayer Pattern Format**

Project:	CHV, BSW

Specifies the format of the Bayer Pattern:

Value	Name	Project
0b	8-bit input at a 8-bit stride	
1b	12 or 10-bit input at a 16-bit stride. Valid data is in the MSBs	CHV, BSW

23:21 **Reserved**

Project:	
Format:	MBZ

20 **Reserved**

Project:	CHV, BSW
Format:	MBZ

19:3 **Surface Pitch**

Format:	U17 pitch in (Bytes - 1)

This field specifies the surface pitch in (#Bytes - 1):

Value	Name	Description
[63, 131071]	For other linear surfaces	[64B, 128KB]
[511, 131071]	For X-tiled surface	[512B, 128KB] = [1tile, 256 tiles]
[127, 131071]	For Y-tiled surfaces	[128B,128KB] = [1 tile, 1024 tiles]

VEBOX_SURFACE_STATE

		Programming Notes For tiled surfaces, the pitch must be a multiple of the tile width. For linear surfaces, the pitch must be a multiple of 64. If Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.
	2	**Half Pitch for Chroma** Format: Enable This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the **Surface Pitch** field. This field is only used for PLANAR surface formats. **Programming Notes** Must be programmed to Zero always as this field is not used
	1	**Tiled Surface** Format: Boolean This field specifies whether the surface is tiled. **Value** / **Name** / **Description** 1 / True / Tiled 0 / False / Linear **Programming Notes** Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled surfaces can only be mapped to Main Memory. The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
	0	**Tile Walk** Format: 3D_TileWalk This field specifies the type of memory tiling (XMajor or YMajor) employed to tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions. This field is ignored when the surface is linear. **Value** / **Name** 0 / TILEWALK_XMAJOR 1 / TILEWALK_YMAJOR **Programming Notes** The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this bit.
4	31:29	**Reserved**
	28:16	**X Offset for U** Format: U13 Pixel Offset This field must be zero for the VEBOX surface formats

VEBOX_SURFACE_STATE

	15	**Reserved** Format: MBZ
	14:0	**Y Offset for U** Format: U15 Row Offset This field specifies the veritical offset in rows from the **Surface Base Address** to the start (origin) of the U(Cb) plane or the interleaved UV plane if **Interleave Chroma** is enabled. This field is only used for PLANAR surface formats. **Programming Notes** This field must indicate an even number (bit 0 = 0). This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line) For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for U should be an integral multiple of the Tile height of the Luma plane
5	31:29	**Reserved** Format: MBZ
	28:16	**X Offset for V** Format: U13 Pixel Offset This field must be zero for the VEBOX surface formats.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for V** Format: U15 Row Offset This field specifies the veritical offset in rows from the **Surface Base Address** to the start (origin) of the V(Cr) plane. This field is only used for PLANAR surface formats with **Interleave Chroma** disabled. **Programming Notes** This field must indicate an even number (bit 0 = 0). This field must be evenly divisible by 4 for Tile-Y surfaces (so the offset points to the start of a cache line). For Planar formats, if the surface is in YS or YF tile modes, the Y Offset for V should be an integral multiple of the Tile height of the Luma plane
6..7	31:0	**Reserved** Project: CHV, BSW Format: MBZ

Wait Notification

wait - Wait Notification

Project:	CHV, BSW
Source:	Eulsa
Length Bias:	4

The wait instruction evaluates the value of the notification count register nreg. If nreg is zero, thread execution is suspended and the thread is put in 'wait_for_notification' state. If nreg is not zero (i.e., one or more notifications have been received), nreg is decremented by one and the thread continues executing on the next instruction. If a thread is in the 'wait_for_notification' state, when a notification arrives, the notification count register is incremented by one. As the notification count register becomes nonzero, the thread wakes up to continue execution and at the same time the notification register is decremented by one. If only one notification arrived, the notification register value becomes zero. However, during the above mentioned time period, it is possible that more notifications may arrive, making the notification register nonzero again. When multiple notifications are received, software must use wait instructions to decrement notification count registers for each notification. Notification register n0.0:ud is for thread to thread communication (via the Message Gateway shared function) and n0.1:ud for host to thread communication (through MMIO registers). See the Message Gateway chapter for thread-thread communication.

Format: wait (exec_size) nreg

Restriction
src0 and dst must be n0.0, n0.1, or n0.2.
Execution size must be 1 as the notification registers are scalar.
Predication is not allowed.
Two back-to-back wait instructions are not allowed. At minimum, a nop instruction must be inserted between two wait instructions

Syntax
wait (1) n#

Pseudocode
N/A

Predication	Conditional Modifier	Saturation	Source Modifier
N	N	N	N

Src Types	Dst Types
UD	UD

DWord	Bit	Description
0	127:64	**Sources** Exists If: ([Operand Control][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**Sources** Exists If: ([Operand Control][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

While

while - While

Project: CHV, BSW
Source: Eulsa
Length Bias: 4

Description

The while instruction marks the end of a do-while block. The instruction first evaluates the loop termination condition for each channel based on the current channel enables and the predication flags specified in the instruction. If any channel has not terminated, a branch is taken to a destination address specified in the instruction, and the loop continues for those channels. Otherwise, execution continues to the next instruction.ld point to the first instruction with the do label of the do-while block of code. It should be a negative number for the backward referencing. In GEN binary, JIP is at location dst and must be of type W (signed word integer). If SPF is ON, none of the PcIP are updated.

The following table describes the 32-bit jump target offset JIP. JIP is a signed 32-bit number, added to IP pre-increment, and should point to the first instruction with the do label of the do-while block of code. It should be a negative number for the backward referencing. In GEN binary, JIP is at location src1 and must be of type D (signed dword integer).

Format: [(pred)] while (exec_size) JIP

Restriction

The execution size must be the same for the while instruction and any break and cont instructions of the same code block.

Syntax	Project
[(pred)] while (exec_size) imm32	CHV, BSW

Pseudocode

```
Evaluate(WrEn); for ( n = 0; n < 32; n++ ) { if (WrEn.chan[n] ) { PcIP[n] = IP + JIP; } else { PcIP[n] = IP + 1; } } if ( | PMask == 1 ) { // any enabled channel true Jump(IP + JIP); }
```

Predication	Conditional Modifier	Saturation	Source Modifier
Y	N	N	N

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW

while - While

	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

XY_COLOR_BLT

XY_COLOR_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

COLOR_BLT is the simplest BLT operation. It performs a color fill to the destination (with a possible ROP). The only operand is the destination operand which is written dependent on the raster operation. The solid pattern color is stored in the pattern background register.
This instruction is optimized to run at the maximum memory write bandwidth.
The typical (and fastest) Raster operation code = F0 which performs a copy of the pattern background register to the destination.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 50h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value: 1xb — Name: Write Alpha Channel Value: x1b — Name: Write RGB Channel
	19:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value: 0b — Name: Tiling Disabled (Linear Blit) Value: 1b — Name: Tiling Enabled — Description: Tile-X or Tile-Y
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 05h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value — Name

XY_COLOR_BLT

		0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR16	31:0	**Solid Pattern Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_FULL_BLT

XY_FULL_BLT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and pattern operands are the same bit width as the destination operand.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 55h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved**
	15	**Src Tiling Enable** Value / Name / Description: 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled [CHV, BSW]: Tile-X or Tile-Y.
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.

XY_FULL_BLT

	11	**Dest Tiling Enable** Value / Name / Description: 0b – Tiling Disabled (Linear Blit) 1b – Tiling Enabled – [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 0Ah
1 BR13	31	**Reserved**
	30	**Clipping Enabled** Value / Name: 0b – Disabled 1b – Enabled
	29:26	**Reserved**
	25:24	**Color Depth** Value / Name: 00b – 8 Bit Color 01b – 16 Bit Color(565) 10b – 16 Bit Color(1555) 11b – 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.

XY_FULL_BLT		
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR11	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8 BR12	31:0	**Source Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL(64byte) aligned.
9 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
10 BR15	31:0	**Pattern Base Address** Format: GraphicsAddress[31:0] (28:06 are implemented) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. The pattern data must be located in linear memory. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
11 BR29	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Pattern Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_FULL_IMMEDIATE_PATTERN_BLT

XY_FULL_IMMEDIATE_PATTERN_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source and immediate pattern operands are the same bit width as the destination operand. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64 DWs) for 8, 16, and 32 bpp color patterns. DWL indicates the total number of Dwords of immediate data.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 74h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** 0b Tiling Disabled (Linear)

XY_FULL_IMMEDIATE_PATTERN_BLT

		1b: Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1 08 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)**

XY_FULL_IMMEDIATE_PATTERN_BLT

		16 bit signed number.
4 BR9	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR11	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8 BR12	31:0	**Source Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Src Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
9 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
10..n	31:0	**Immediate Data 0**

XY_FULL_MONO_PATTERN_BLT

XY_FULL_MONO_PATTERN_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The pattern operand is monochrome and the source operand is the same bit width as the destination operand.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 57h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved**

XY_FULL_MONO_PATTERN_BLT

	15	**Src Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Dest Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vectical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Value / Name 0Ch /
1 BR13	31	**Solid Pattern Select** Value / Name 0 / No Solid Pattern 1 / Solid Pattern
	30	**Clipping Enabled** Value / Name 0b / Disabled 1b / Enabled
	29	**Reserved** Format: / MBZ
	28:27	**Mono Source Transparency Mode** Value / Name 0 / Use Background 1 / Transparency Enabled
	26	**Reserved**
	25:24	**Color Depth** Value / Name 00b / 8 Bit Color 01b / 16 Bit Color(565) 10b / 16 Bit Color(1555) 11b / 32 Bit Color

XY_FULL_MONO_PATTERN_BLT		
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned and signed) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
7 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
8 BR12	31:0	**Source Address** Format: GraphicsAddress[31:0] (base address of the source surface: X=0, Y=0). Lower 32bits of the 48bit addressing. When Src Tiling is enabled (Bit 15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.

XY_FULL_MONO_PATTERN_BLT		
9 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
10 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
11 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
12 BR20	31:0	**Pattern Data 0** (least significant DW)
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The full BLT provides the ability to specify all 3 operands: destination, source, and pattern. The pattern and source operands are monochrome.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.
All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the source bit is 1, then the pattern foreground color is used in the ROP operation. The monochrome source transparency mode works identical to the pattern transparency mode.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Setting both Solid Pattern Select =1 and Mono Pattern Transparency = 1 is mutually exclusive. The device implementation results in NO PIXELs DRAWN.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 58h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y = 0.
	7:0	**DWord Length** Value / Name 0Ch /
1 BR13	31	**Solid Pattern Select** Value / Name 0 / No Solid Pattern 1 / Solid Pattern
	30	**Clipping Enabled** Value / Name 0b / Disabled 1b / Enabled
	29	**Mono Source Transparency Mode** Value / Name 0 / Use Background 1 / Transparency Enabled
	28	**Mono Pattern Transparency Mode** Value / Name 0 / Use Background 1 / Transparency Enabled
	27:26	**Reserved**
	25:24	**Color Depth** Value / Name 00b / 8 Bit Color 01b / 16 Bit Color(565) 10b / 16 Bit Color(1555) 11b / 32 Bit Color

XY_FULL_MONO_PATTERN_MONO_SRC_BLT		
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes.When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR12	31:0	**Mono Source Address** Format: GraphicsAddress[31:0] (address corresponds to DST X1, Y1) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. This Monosource Base Address programmed, must always be Cache Line (64byte) aligned.
7 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Mono Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_FULL_MONO_PATTERN_MONO_SRC_BLT		
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
11 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
12 BR20	31:0	**Pattern Data 0** (least significant DW)
13 BR21	31:0	**Pattern Data 1** (most significant DW)

XY_FULL_MONO_SRC_BLT

XY_FULL_MONO_SRC_BLT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is monochrome and the pattern operand is the same bit width as the destination.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation.
All non-text and non-immediate monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the Destination X1 coordinate.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Negative Stride (= Pitch) is NOT ALLOWED

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 56h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)

XY_FULL_MONO_SRC_BLT

	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y = 0.
	7:0	**DWord Length** Value: 0Ah / Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Mono Source Transparency Mode** 0: Use Background 1: Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.

XY_FULL_MONO_SRC_BLT

BR23	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR12	31:0	**Mono Source Address** Format: GraphicsAddress[31:0] (address corresponds to DST X1, Y1) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. This Monosource Base Address programmed, must always be Cache Line (64byte) aligned.
7 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Mono Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10 BR15	31:0	**Pattern Base Address** Format: GraphicsAddress[31:0] (28:06 are implemented) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. The pattern data must be located in linear memory. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.

XY_FULL_MONO_SRC_BLT		
11 BR29	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Pattern Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

Project:	CHV, BSW
Source:	BlitterCS
Length Bias:	2

The full BLT is the most comprehensive BLT instruction. It provides the ability to specify all 3 operands: destination, source, and pattern. The source operand is a monochrome and the immediate pattern operand is the same bit width as the destination. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns. The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. All non-text monochrome sources are word aligned. At the end of a scan line the monochrome source, the remaining bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates which bit position within the first byte should be used as the first source pixel which corresponds to the destination X1 coordinate. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation. The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8. Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 75h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** (pixel of the scan line to start on corresponding to DST X=0)

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

	11	**Tiling Enable** Value / Name / Description: 0b – Tiling Disabled (Linear Blit) 1b – Tiling Enabled – [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Starting scan line of the 8x8 pattern corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1 08 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value / Name: 0b – Disabled 1b – Enabled
	29	**Mono Source Transparency Mode** Value / Name: 0 – Use Background 1 – Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value / Name: 00b – 8 Bit Color 01b – 16 Bit Color(565) 10b – 16 Bit Color(1555) 11b – 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT		
BR23	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR12	31:0	**Mono Source Address** Format: GraphicsAddress[31:0] (address corresponds to DST X1, Y1) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. This Monosource Base Address programmed, must always be Cache Line (64byte) aligned.
7 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Mono Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
10..n	31:0	**Immediate Data**

XY_MONO_PAT_BLT

XY_MONO_PAT_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

MONO_PAT_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is loaded from the instruction stream.
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 52h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.

XY_MONO_PAT_BLT

	7:0	**DWord Length** Value: 08h Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Reserved** Format: MBZ
	28	**Mono Pattern Transparency Mode** 0: Use Background 1: Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0]

XY_MONO_PAT_BLT

		Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8 BR20	31:0	**Pattern Data 0**
9 BR21	31:0	**Pattern Data 1**

XY_MONO_PAT_FIXED_BLT

XY_MONO_PAT_FIXED_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

MONO_PAT_FIXED_BLT is used when we have no source and the monochrome pattern is not trivial (is not a solid color only). The monochrome pattern is one of 10 fixed patterns described below. The pattern seeds can still be used with the fixed patterns, creating even more fixed patterns. This eliminates 2 doublewords compared to the XY_MONO_PAT_BLT command packet.

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

The monochrome pattern transparency mode indicates whether to use the pattern background color or de-assert the write enables when the bit in the pattern is 0. When the pattern bit is 1, then the pattern foreground color is used in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 59h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19	**Reserved** Format: MBZ
	18:15	**Fixed Pattern** 0000b HS_HORIZONTAL 0001b HS_VERTICAL 0010b HS_FDIAGONAL 0011b HS_BDIAGONAL

XY_MONO_PAT_FIXED_BLT

		0100b: HS_CROSS 0101b: HS_DIAGCROSS 0110b: Reserved 0111b: Reserved 1000b: Screen Door 1001b: SD Wide 1010b: Walking Bit (one) 1011b: Walking Zero 1100b: Reserved 1101b: Reserved 1110b: Reserved 1111b: Reserved
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** 06h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Reserved** Format: MBZ
	28	**Mono Pattern Transparency Mode** 0: Use Background 1: Transparency Enabled
	27:26	**Reserved**

XY_MONO_PAT_FIXED_BLT

	25:24	**Color Depth** Value / Name: 00b – 8 Bit Color 01b – 16 Bit Color(565) 10b – 16 Bit Color(1555) 11b – 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR16	31:0	**Pattern Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR17	31:0	**Pattern Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_MONO_SRC_COPY_BLT

XY_MONO_SRC_COPY_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a monochrome source copy where the only operands involved is a monochrome source and destination. The source and destination operands cannot overlap therefore the X and Y directions are always forward.

All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.

The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation. Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 54h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ

XY_MONO_SRC_COPY_BLT

	7:0	**DWord Length** Value: 08h Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29	**Mono Source Transparency Mode** 0: Use Background 1: Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.

XY_MONO_SRC_COPY_BLT

5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR12	31:0	**Mono Source Address** Format: GraphicsAddress[31:0] (address corresponds to DST X1, Y1) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. This Monosource Base Address programmed, must always be Cache Line (64byte) aligned.
7 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Mono Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
8 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
9 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]

XY_MONO_SRC_COPY_IMMEDIATE_BLT

XY_MONO_SRC_COPY_IMMEDIATE_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

This instruction allows the Driver to send monochrome data through the instruction stream, eliminating the read latency of the source during command execution.
The IMMEDIATE_BLT data MUST transfer an even number of doublewords and the exact number of quadwords. DWL indicates the total number of Dwords of immediate data.
All non-text monochrome sources are word aligned. At the end of a scan line of monochrome source, all bits until the next word boundary are ignored. The Monochrome source data bit position field [2:0] indicates the bit position within the first byte of the scan line that should be used as the first source pixel which corresponds to the destination X1 coordinate.
The monochrome source transparency mode indicates whether to use the source background color or de-assert the write enables when the bit in the source is 0. When the source bit is 1, then the source foreground color is used in the ROP operation. The ROP value chosen must involve source and no pattern data in the ROP operation.
The monochrome source data supplied corresponds to the Destination X1 and Y1 coordinates.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 71h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Monochrome source data bit position of the first pixel within a byte per scan line.**
	16:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b Tiling Disabled (Linear) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.

XY_MONO_SRC_COPY_IMMEDIATE_BLT

	10:8	**Reserved**
	7:0	**DWord Length** Default Value: 06h Excludes DWORD 0,1 06 + DWL = (Number of Immediate double words)h
1 BR13	31	**Reserved**
	30	**Clipping Enabled** Value: 0b — Disabled Value: 1b — Enabled
	29	**Mono Source Transparency Mode** Value: 0b — Transparency Enabled Value: 1b — Use Background
	28:26	**Reserved**
	25:24	**Color Depth** Value: 00b — 8 Bit Color Value: 01b — 16 Bit Color(565) Value: 10b — 16 Bit Color(1555) Value: 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.

XY_MONO_SRC_COPY_IMMEDIATE_BLT		
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR18	31:0	**Source Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
7 BR19	31:0	**Source Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0]
8..n	31:0	**Immediate Data**

XY_PAT_BLT

XY_PAT_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
If clipping is enabled, all scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 51h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved**
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 06h
1	31	**Reserved**

XY_PAT_BLT

BR13, 30 — Clipping Enabled

Value	Name
0b	Disabled
1b	Enabled

BR13, 29:26 — Reserved

Format:	MBZ

BR13, 25:24 — Color Depth

Value	Name
00b	8 Bit Color
01b	16 Bit Color(565)
10b	16 Bit Color(1555)
11b	32 Bit Color

BR13, 23:16 — Raster Operation

BR13, 15:0 — Destination Pitch in DWords

2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).

2, BR22, 31:16 — Destination Y1 Coordinate (Top)

16 bit signed number.

2, BR22, 15:0 — Destination X1 Coordinate (Left)

16 bit signed number.

3, BR23, 31:16 — Destination Y2 Coordinate (Bottom)

16 bit signed number.

3, BR23, 15:0 — Destination X2 Coordinate (Right)

16 bit signed number.

4, BR09, 31:0 — Destination Base Address

Format:	GraphicsAddress[31:0]

Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.

5, BR27, 31:16 — Reserved

Format:	MBZ

Should be programmed all 0's for 48bit addressing.

5, BR27, 15:0 — Destination Base Address High

Format:	GraphicsAddress[47:32]

Should be programmed with the upper 16bits of the 48bit addressing.

<table>
<tr><th colspan="3">XY_PAT_BLT</th></tr>
<tr><td>6

BR15</td><td>31:0</td><td>Pattern Base Address<table><tr><td>Format:</td><td>GraphicsAddress[31:0]</td></tr></table>(28:06 are implemented) (Note no NPO2 change here). Lower 32bits of the 48bit addressing.
The pattern data must be located in linear memory.
The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.</td></tr>
<tr><td rowspan="2">7

BR29</td><td>31:16</td><td>Reserved<table><tr><td>Format:</td><td>MBZ</td></tr></table>Should be programmed all 0's for 48bit addressing.</td></tr>
<tr><td>15:0</td><td>Pattern Base Address High<table><tr><td>Format:</td><td>GraphicsAddress[47:32]</td></tr></table>Should be programmed with the upper 16bits of the 48bit addressing.</td></tr>
</table>

XY_PAT_BLT_IMMEDIATE

XY_PAT_BLT_IMMEDIATE

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 72h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:15	**Reserved** Format: MBZ
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length**

XY_PAT_BLT_IMMEDIATE

		Default Value: 04h Excludes DWORD 0,1 04 + DWL = (Number of Immediate double)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value: 0b — Name: Disabled Value: 1b — Name: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value: 00b — Name: 8 Bit Color Value: 01b — Name: 16 Bit Color(565) Value: 10b — Name: 16 Bit Color(1555) Value: 11b — Name: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.

XY_PAT_BLT_IMMEDIATE		
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6..n	31:0	**Immediate Data**

XY_PAT_CHROMA_BLT

XY_PAT_CHROMA_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

PAT_BLT is used when there is no source and the color pattern is not trivial (is not a solid color only).
All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 76h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved**
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 08h Excludes DWORD 0,1

XY_PAT_CHROMA_BLT

1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_PAT_CHROMA_BLT		
6 BR15	31:0	**Pattern Base Address** Format: GraphicsAddress[31:0] (28:06 are implemented) (Note no NPO2 change here). Lower 32bits of the 48bit addressing. The pattern data must be located in linear memory. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
7 BR29	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Pattern Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
8 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
9 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_PAT_CHROMA_BLT_IMMEDIATE

XY_PAT_CHROMA_BLT_IMMEDIATE

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

PAT_BLT_IMMEDIATE is used when there is no source and the color pattern is not trivial (is not a solid color only) and the pattern is pulled through the command stream. The immediate data sizes are 64 bytes (16 DWs), 128 bytes (32 DWs), or 256 (64DWs) for 8, 16, and 32 bpp color patterns.

DWL indicates the total number of Dwords of immediate data. All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.

The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 77h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key) - Dst Chroma-key modes ONLY (SRC ILLEGAL)
	16:15	**Reserved**
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.

XY_PAT_CHROMA_BLT_IMMEDIATE

	7:0	**DWord Length** Default Value: 06h Excludes DWORD 0,1 06 + DWL = (Number of Immediate double)h
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.

XY_PAT_CHROMA_BLT_IMMEDIATE		
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
7 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)
8..n	31:0	**Immediate Data**

XY_PIXEL_BLT

XY_PIXEL_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

The Destination X coordinate and Destination Y coordinate is compared with the ClipRect registers. If it is within all 4 comparisons, then the pixel supplied in the XY_SETUP_BLT instruction is written with the raster operation to (Destination Y Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).
ROP field must specify pattern or fill with 0's or 1's. There is no source operand.
Negative Stride (= Pitch) specified in the Setup command is Not Allowed

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 24h Format: Opcode
	21:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 00h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.

XY_SCANLINES_BLT

XY_SCANLINES_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

All scan lines and pixels that fall within the ClipRect Y and X coordinates are written. Only pixels within the ClipRectX coordinates and the Destination X coordinates are written using the raster operation.
The Pattern Seeds correspond to Destination X = 0 (horizontal) and Y = 0 (vertical). The alignment is relative to the destination coordinates. The pixel of the pattern used / scan line is the (destination X coordinate + horizontal seed) modulo 8. The scan line of the pattern used is the (destination Y coordinate + vertical seed) modulo 8.
Solid pattern should use the XY_SETUP_MONO_PATTERN_SL_BLT instruction.
ROP field must specify pattern or fill with 0's or 1's. There is no source operand.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 25h Format: Opcode
	21:15	**Reserved**
	14:12	**Pattern Horizontal Seed** Pixel of the scan line to start on corresponding to DST X=0.
	11	**Tiling Enable** Value / Name / Description 0b / Tiling Disabled (Linear Blit) / 1b / Tiling Enabled / [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Pattern Vertical Seed** Scan line of the 8x8 pattern to start on corresponding to DST Y=0.
	7:0	**DWord Length** Default Value: 01h
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.

XY_SETUP_BLT

XY_SETUP_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

This setup instruction supplies common setup information including clipping coordinates used by the XY commands: XY_PIXEL_BLT, XY_SCANLINE_BLT, XY_TEXT_BLT, and XY_TEXT_BLT_IMMEDIATE.
These are the only instructions that require that state be saved between instructions other than the Clipping parameters. There are 5 dedicated registers to contain the state for the 3 setup BLT instructions (XY_SETUP_BLT, XY_SETUP_MONO_PATTERN_SL_BLT, and XY_SETUP_CLIP_BLT. All other BLTs use a temporary version of these. The 5 double word registers are: DW1 (Setup Control), DW6 (Setup Foreground color), DW5 (Setup Background color), DW7 (Setup Pattern address), and DW4 (Setup Destination Base Address).

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 01h Format: Opcode
	21:20	**32 bpp Byte Mask** Value: 1xb — Name: Write Alpha Channel Value: x1b — Name: Write RGB Channel
	19:12	**Reserved**
	11	**Tiling Enable** Value: 0b — Name: Tiling Disabled (Linear Blit) Value: 1b — Name: Tiling Enabled (Tile-X or Tile-Y)
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Reserved**
	30	**Clipping Enabled** Value: 0b — Name: Disabled Value: 1b — Name: Enabled

XY_SETUP_BLT

	29	**Mono Source Transparency Mode** Value: 0b — Use Background Value: 1b — Transparency Enabled
	28:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b — 8 Bit Color 01b — 16 Bit Color(565) 10b — 16 Bit Color(1555) 11b — 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
3 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)
4 BR09	31:0	**Setup Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Setup Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_SETUP_BLT

6 BR05	31:0	**Setup Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
7 BR06	31:0	**Setup Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)
8 BR07	31:0	**Setup Pattern Base Address for Color Pattern** Format: GraphicsAddress[31:0] (26:06 are implemented) (SLB only) (Note no NPO2 change here). The pattern data must be located in linear memory. Lower 32bits of the 48bit addressing. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
9 BR30	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Setup Pattern Base Address for Color Pattern High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_SETUP_CLIP_BLT

XY_SETUP_CLIP_BLT		
Project:	CHV, BSW	
Source:	BlitterCS	
Length Bias:	2	
This command is used to only change the clip coordinate registers. These are the same clipping registers as the Setup clipping registers above.		
DWord	**Bit**	**Description**
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 03h Format: Opcode
	21:12	**Reserved** Format: MBZ
	11	**Tiling Enable** Value / Name 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 01h
1 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
2 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)

XY_SETUP_MONO_PATTERN_SL_BLT

XY_SETUP_MONO_PATTERN_SL_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

This setup instruction supplies common setup information including clipping coordinates used exclusively with the following instruction: XY_SCANLINE_BLT (SLB) - 1 scan line of monochrome pattern and destination are the only operands allowed.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 11h Format: Opcode
	21:20	**32 bpp Byte Mask** 1xb: Write Alpha Channel x1b: Write RGB Channel
	19:12	**Reserved**
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled (Tile-X or Tile-Y
	10:8	**Reserved**
	7:0	**DWord Length** Default Value: 08h
1 BR01	31	**Solid Pattern Select** (SLB and Pixel only) 0: No Solid Pattern 1: Solid Pattern
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled

XY_SETUP_MONO_PATTERN_SL_BLT

	29	**Reserved** Format: MBZ
	28	**Mono Pattern Transparency Mode** 0b: Use Background 1b: Transparency Enabled
	27:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement (Negative Pitch Not allowed for Pixel nor Text) For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR24	31:16	**ClipRect Y1 Coordinate (Top)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X1 Coordinate (Left)** (14:00 = 15 bit positive number)
3 BR25	31:16	**ClipRect Y2 Coordinate (Bottom)** (30:16 = 15 bit positive number)
	15:0	**ClipRect X2 Coordinate (Right)** (14:00 = 15 bit positive number)
4 BR09	31:0	**Setup Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Setup Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_SETUP_MONO_PATTERN_SL_BLT		
6 BR05	31:0	**Setup Background Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] All
7 BR06	31:0	**Setup Foreground Color** 8 bit = [7:0], 16 bit = [15:0], 32 bit = [31:0] (SLB and TB only)
8 BR20	31:0	**DW0 (least significant) for a Monochrome Pattern**
9 BR21	31:0	**DW1 (most significant) for a Monochrome Pattern**

XY_SRC_COPY_BLT

XY_SRC_COPY_BLT

Project: CHV, BSW

Source: BlitterCS

Length Bias: 2

This BLT instruction performs a color source copy where the only operands involved is a color source and destination of the same bit width.

The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.

The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 53h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name: 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:16	**Reserved**
	15	**Src Tiling Enable** 0b Tiling Disabled (Linear) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	14:12	**Reserved**
	11	**Dest Tiling Enable** 0b Tiling Disabled (Linear Blit) 1b Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.

XY_SRC_COPY_BLT

	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Value: 08h Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** 0b: Disabled 1b: Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** 00b: 8 Bit Color 01b: 16 Bit Color(565) 10b: 16 Bit Color(1555) 11b: 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address should be CL (64byte) aligned.

XY_SRC_COPY_BLT		
5 BR27	31:16	**Reserved** Format: MBZ Must be all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] The upper 16bits of the 48-bit address.
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
8 BR12	31:0	**Source Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Src Tiling is enabled (Bit_15 enabled), this address must be 4KB-aligned. When Tiling is not enabled, this address should be CL (64byte) aligned.
9 BR28	31:16	**Reserved** Format: MBZ Must be all 0's for 48-bit addressing.
	15:0	**Source Base Address High** Format: GraphicsAddress[47:32] The upper 16 bits of the 48-bit address.

XY_SRC_COPY_CHROMA_BLT

XY_SRC_COPY_CHROMA_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

This BLT instruction performs a color source copy with chroma-keying where the only operands involved is a color source and destination of the same bit width.
The source and destination operands may overlap, which means that the X and Y directions can be either forward or backwards. The BLT Engine takes care of all situations. The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses start at Destination Y2 with the corresponding source scan line and the strides are subtracted for every scan line access.
The ROP value chosen must involve source and no pattern data in the ROP operation.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 73h Format: Opcode
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value / Name 00b **[Default]** 1xb Write Alpha Channel x1b Write RGB Channel
	19:17	**Transparency Range Mode** (chroma-key)
	16	**Reserved** Format: MBZ
	15	**Src Tiling Enable** Value / Name / Description 0b Tiling Disabled (Linear) 1b Tiling Enabled [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	14:12	**Reserved** Format: MBZ

XY_SRC_COPY_CHROMA_BLT

	11	**Dest Tiling Enable** Value / Name / Description: 0b – Tiling Disabled (Linear Blit) 1b – Tiling Enabled – [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Value: 0Ah – Name:
1 BR13	31	**Reserved** Format: MBZ
	30	**Clipping Enabled** Value / Name: 0b – Disabled 1b – Enabled
	29:26	**Reserved** Format: MBZ
	25:24	**Color Depth** Value / Name: 00b – 8 Bit Color 01b – 16 Bit Color(565) 10b – 16 Bit Color(1555) 11b – 32 Bit Color
	23:16	**Raster Operation**
	15:0	**Destination Pitch in DWords** 2's complement For Tiled surfaces (bit_11 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Y and can be upto 128Kbytes (or 32KDwords).
2 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
3 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
4 BR09	31:0	**Destination Base Address** Format: GraphicsAddress[31:0]

XY_SRC_COPY_CHROMA_BLT

		Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_11 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
5 BR27	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Destination Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
6 BR26	31:16	**Source Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Source X1 Coordinate (Left)** 16 bit signed number.
7 BR11	31:16	**Reserved** Format: MBZ
	15:0	**Source Pitch (double word aligned) and in DWords** 2's complement. For Tiled Src (bit 15 enabled) this pitch is of 512Byte granularity for Tile-X, 128B granularity for Tile-Yand can be upto 128Kbytes (or 32KDwords).
8 BR12	31:0	**Source Base Address** Format: GraphicsAddress[31:0] Base address of the destination surface: X=0, Y=0. Lower 32bits of the 48bit addressing. When Tiling is enabled (Bit_15 enabled), this address is limited to 4Kbytes. When Tiling is not enabled, this address should be CL (64byte) aligned.
9 BR28	31:16	**Reserved** Format: MBZ Should be programmed all 0's for 48bit addressing.
	15:0	**Source Base Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.
10 BR18	31:0	**Transparency Color Low** (Chroma-key Low = Pixel Greater or Equal)
11 BR19	31:0	**Transparency Color High** (Chroma-key High = Pixel Less or Equal)

XY_TEXT_BLT

XY_TEXT_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

All source scan lines and pixels that fall within the ClipRect Y and X coordinates are written. The source address corresponds to Destination X1 and Y1 coordinate.
Text is either bit or byte packed. Bit packed means that the next scan line starts 1 pixel after the end of the current scan line with no bit padding. Byte packed means that the next scan line starts on the first bit of the next byte boundary after the last bit of the current line.
Source expansion color registers are always in the SETUP_BLT.
Negative Stride (= Pitch) is NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 26h Format: Opcode
	21:17	**Reserved** Format: MBZ
	16	**Bit / Byte Packed** Byte packed is for the NT driver. Value: 0 — Name: Bit Value: 1 — Name: Byte
	15:12	**Reserved**
	11	**Tiling Enable** 0b — Tiling Disabled (Linear Blit) 1b — Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Reserved** Format: MBZ
	7:0	**DWord Length** Default Value: 03h
1	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.

XY_TEXT_BLT

BR22	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3 BR12	31:0	**Source Address** Format: GraphicsAddress[31:0] (address of the first byte on scan line corresponding to Dst X1, Y1). Lower 32bits of the 48bit addressing. (Note no NPO2 change here). Since Text data is Monosource data, the Text source Base Address programmed, must always be Cache Line (64byte) aligned.
4 BR28	31:16	**Reserved** Format: MBZ Should be programmed with all "0"s for 48bit addressing.
	15:0	**Source Address High** Format: GraphicsAddress[47:32] Should be programmed with the upper 16bits of the 48bit addressing.

XY_TEXT_IMMEDIATE_BLT

XY_TEXT_IMMEDIATE_BLT

Project: CHV, BSW
Source: BlitterCS
Length Bias: 2

This instruction allows the Driver to send data through the instruction stream that eliminates the read latency of reading a source from memory.
If an operand is in system cacheable memory and either small or only accessed once, it can be copied directly to the instruction stream versus to graphics accessible memory. The IMMEDIATE_BLT data MUST transfer an even number of doublewords.
The BLT engine will hang if it does not get an even number of doublewords. All source scan lines and pixels that fall within the ClipRect X and Y coordinates are written. The source data corresponds to Destination X1 and Y1 coordinate.
Source expansion color registers are always in the SETUP_BLT. NEGATIVE STRIDE (= PITCH) IS NOT ALLOWED.

DWord	Bit	Description
0 BR00	31:29	**Client** Default Value: 02h 2D Processor Format: Opcode
	28:22	**Instruction Target(Opcode)** Default Value: 31h Format: Opcode
	21:17	**Reserved**
	16	**Bit / Byte Packed** Byte packed is for the NT driver. Value 0: Bit Value 1: Byte
	15:12	**Reserved** Format: MBZ
	11	**Tiling Enable** 0b: Tiling Disabled (Linear Blit) 1b: Tiling Enabled — [CHV, BSW] [CHV, BSW]: Tile-X or Tile-Y.
	10:8	**Reserved**
	7:0	**DWord Length** Default Value: 01h Excludes DWORD 0,1 01 + DWL = (Number of Immediate double words)h

XY_TEXT_IMMEDIATE_BLT		
1 BR22	31:16	**Destination Y1 Coordinate (Top)** 16 bit signed number.
	15:0	**Destination X1 Coordinate (Left)** 16 bit signed number.
2 BR23	31:16	**Destination Y2 Coordinate (Bottom)** 16 bit signed number.
	15:0	**Destination X2 Coordinate (Right)** 16 bit signed number.
3..n	31:0	**Immediate Data**

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 2b: Command Reference: Enumerations

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

3D_Color_Buffer_Blend_Factor 1
3D_Color_Buffer_Blend_Function 2
3D_Compare_Function 3
3D_Logic_Op_Function 4
3D_Prim_Topo_Type 5
3D_Stencil_Operation 7
3D_Vertex_Component_Control 8
AddrMode 9
ChanEn 10
ChanSel 11
Clear Color 12
CondModifier 13
DepCtrl 14
DstType 15
EU_OPCODE 16
ExecSize 19
FC 20
HorzStride 21
PredCtrl 22
QtrCtrl 24
RegFile 26
RepCtrl 27
SFID 28
Shader Channel Select 29
SIMD Mode 30
SrcImmType 31
SrcIndex 32
SrcMod 34
SrcType 35
SURFACE_FORMAT 36
Texture Coordinate Mode 43

ThreadCtrl **44**

VertStride **45**

Width **46**

WRAP_SHORTEST_ENABLE **47**

3D_Color_Buffer_Blend_Factor

3D_Color_Buffer_Blend_Factor	
Source:	PRM
Size (in bits):	5

Value	Name
00h	Reserved
01h	BLENDFACTOR_ONE
02h	BLENDFACTOR_SRC_COLOR
03h	BLENDFACTOR_SRC_ALPHA
04h	BLENDFACTOR_DST_ALPHA
05h	BLENDFACTOR_DST_COLOR
06h	BLENDFACTOR_SRC_ALPHA_SATURATE
07h	BLENDFACTOR_CONST_COLOR
08h	BLENDFACTOR_CONST_ALPHA
09h	BLENDFACTOR_SRC1_COLOR
0Ah	BLENDFACTOR_SRC1_ALPHA
0Bh-10h	Reserved
11h	BLENDFACTOR_ZERO
12h	BLENDFACTOR_INV_SRC_COLOR
13h	BLENDFACTOR_INV_SRC_ALPHA
14h	BLENDFACTOR_INV_DST_ALPHA
15h	BLENDFACTOR_INV_DST_COLOR
16h	Reserved
17h	BLENDFACTOR_INV_CONST_COLOR
18h	BLENDFACTOR_INV_CONST_ALPHA
19h	BLENDFACTOR_INV_SRC1_COLOR
1Ah	BLENDFACTOR_INV_SRC1_ALPHA

3D_Color_Buffer_Blend_Function

3D_Color_Buffer_Blend_Function	
Source:	PRM
Size (in bits):	3

Value	Name	Description
0	BLENDFUNCTION_ADD	BLENDFUNCTION_ADD
1	BLENDFUNCTION_SUBTRACT	BLENDFUNCTION_SUBTRACT
2	BLENDFUNCTION_REVERSE_SUBTRACT	BLENDFUNCTION_REVERSE_SUBTRACT
3	BLENDFUNCTION_MIN	BLENDFUNCTION_MIN
4	BLENDFUNCTION_MAX	BLENDFUNCTION_MAX
5 - 7	Reserved	

3D_Compare_Function

3D_Compare_Function		
Source:	PRM	
Size (in bits):	3	
Value	**Name**	**Description**
0h	COMPAREFUNCTION_ALWAYS	Always pass
1h	COMPAREFUNCTION_NEVER	Never pass
2h	COMPAREFUNCTION_LESS	Pass if the value is less than the reference
3h	COMPAREFUNCTION_EQUAL	Pass if the value is equal to the reference
4h	COMPAREFUNCTION_LEQUAL	Pass if the value is less than or equal to the reference
5h	COMPAREFUNCTION_GREATER	Pass if the value is greater than the reference
6h	COMPAREFUNCTION_NOTEQUAL	Pass if the value is not equal to the reference
7h	COMPAREFUNCTION_GEQUAL	Pass if the value is greater than or equal to the reference

3D_Logic_Op_Function

3D_Logic_Op_Function		
Source:	PRM	
Size (in bits):	4	
Value	**Name**	**Description**
0h	LOGICOP_CLEAR	BLACK; all 0's
1h	LOGICOP_NOR	NOTMERGEPEN; NOT (S OR D)
2h	LOGICOP_AND_INVERTED	MASKNOTPEN; (NOT S) AND D
3h	LOGICOP_COPY_INVERTED	NOTCOPYPEN; NOT S
4h	LOGICOP_AND_REVERSE	MASKPENNOT; S AND NOT D
5h	LOGICOP_INVERT	NOT; NOT D
6h	LOGICOP_XOR	XORPEN; S XOR D
7h	LOGICOP_NAND	NOTMASKPEN; NOT (S AND D)
8h	LOGICOP_AND	MASKPEN; S AND D
9h	LOGICOP_EQUIV	NOTXORPEN; NOT (S XOR D)
Ah	LOGICOP_NOOP	NOP; D
Bh	LOGICOP_OR_INVERTED	MERGENOTPEN; (NOT S) OR D
Ch	LOGICOP_COPY	COPYPEN; S
Dh	LOGICOP_OR_REVERSE	MERGEPENNOT; S OR NOT D
Eh	LOGICOP_OR	MERGEPEN; S OR D
Fh	LOGICOP_SET	WHITE; all 1's

3D_Prim_Topo_Type

3D_Prim_Topo_Type

Project: All
Source: RenderCS
Size (in bits): 6

The following table defines the encoding of the Primitive Topology Type field. See 3D Pipeline for details, programming restrictions, diagrams and a discussion of the basic primitive types.

Value	Name	Description
00h	Reserved	
01h	3DPRIM_POINTLIST	
02h	3DPRIM_LINELIST	
03h	3DPRIM_LINESTRIP	
04h	3DPRIM_TRILIST	
05h	3DPRIM_TRISTRIP	
06h	3DPRIM_TRIFAN	
07h	3DPRIM_QUADLIST	The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline.
08h	3DPRIM_QUADSTRIP	The QUADSTRIP topology is converted to POLYGON topology at the beginning of the 3D pipeline.
09h	3DPRIM_LINELIST_ADJ	
0Ah	3DPRIM_LISTSTRIP_ADJ	
0Bh	3DPRIM_TRILIST_ADJ	
0Ch	3DPRIM_TRISTRIP_ADJ	
0Dh	3DPRIM_TRISTRIP_REVERSE	
0Eh	3DPRIM_POLYGON	
0Fh	3DPRIM_RECTLIST	
10h	3DPRIM_LINELOOP	The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline.
11h	3DPRIM_POINTLIST _BF	
12h	3DPRIM_LINESTRIP_CONT	
13h	3DPRIM_LINESTRIP_BF	
14h	3DPRIM_LINESTRIP_CONT_BF	
15h	Reserved	Reserved for HW use as TRISTRIP_ADJ_REV
16h	3DPRIM_TRIFAN_NOSTIPPLE	
17h	Reserved	Reserved for HW use as POLYGON_CONT
18h	Reserved	Reserved for HW use as LINESTRIP_ADJ_CONT

3D_Prim_Topo_Type

19h	Reserved	
1Ah-1Fh	Reserved	
20h	3DPRIM_PATCHLIST_1	List of 1-vertex patches
21h	3DPRIM_PATCHLIST_2	
22h	3DPRIM_PATCHLIST_3	
23h	3DPRIM_PATCHLIST_4	
24h	3DPRIM_PATCHLIST_5	
25h	3DPRIM_PATCHLIST_6	
26h	3DPRIM_PATCHLIST_7	
27h	3DPRIM_PATCHLIST_8	
28h	3DPRIM_PATCHLIST_9	
29h	3DPRIM_PATCHLIST_10	
2ah	3DPRIM_PATCHLIST_11	
2bh	3DPRIM_PATCHLIST_12	
2ch	3DPRIM_PATCHLIST_13	
2dh	3DPRIM_PATCHLIST_14	
2eh	3DPRIM_PATCHLIST_15	
2fh	3DPRIM_PATCHLIST_16	
30h	3DPRIM_PATCHLIST_17	
31h	3DPRIM_PATCHLIST_18	
32h	3DPRIM_PATCHLIST_19	
33h	3DPRIM_PATCHLIST_20	
34h	3DPRIM_PATCHLIST_21	
35h	3DPRIM_PATCHLIST_22	
36h	3DPRIM_PATCHLIST_23	
37h	3DPRIM_PATCHLIST_24	
38h	3DPRIM_PATCHLIST_25	
39h	3DPRIM_PATCHLIST_26	
3ah	3DPRIM_PATCHLIST_27	
3bh	3DPRIM_PATCHLIST_28	
3ch	3DPRIM_PATCHLIST_29	
3dh	3DPRIM_PATCHLIST_30	
3eh	3DPRIM_PATCHLIST_31	
3Fh	3DPRIM_PATCHLIST_32	List of 32-vertex patches

3D_Stencil_Operation

3D_Stencil_Operation	
Project:	CHV, BSW
Source:	RenderCS
Size (in bits):	3

Value	Name
0	STENCILOP_KEEP
1	STENCILOP_ZERO
2	STENCILOP_REPLACE
3	STENCILOP_INCRSAT
4	STENCILOP_DECRSAT
5	STENCILOP_INCR
6	STENCILOP_DECR
7	STENCILOP_INVERT

3D_Vertex_Component_Control

3D_Vertex_Component_Control

Project: All
Source: RenderCS
Size (in bits): 3

Value	Name	Description
0	VFCOMP_NOSTORE	Don't store this component. (Not valid for Component 0, but can be used for Component 1-3). Once this setting is used for a component, all higher-numbered components (if any) MUST also use this setting. (I.e., no holes within any particular vertex element). VFCOMP_NOSTORE will not store a component if the SourceElementFormat is R64_PASSTHRU or R64G64_PASSTHRU and it is used on component 2 and 3 else 0 will be stored.
1	VFCOMP_STORE_SRC	Store corresponding component from format-converted source element. Storing a component that is not included in the Source Element Format results in an UNPREDICTABLE value being stored. Software should used the STORE_0 or STORE_1 encoding to supply default components.Within a VERTEX_ELEMENT_STATE structure, if a Component Control field is set to something other than VFCOMP_STORE_SRC, no higher-numbered Component Control fields may be set to VFCOMP_STORE_SRC. In other words, only trailing components can be set to something other than VFCOMP_STORE_SRC.
2	VFCOMP_STORE_0	Store 0 (interpreted as 0.0f if accessed as a float value)
3	VFCOMP_STORE_1_FP	Store 1.0f
4	VFCOMP_STORE_1_INT	Store 0x1
5-6	-	Reserved
7	VFCOMP_STORE_PID	Store Primitive ID (as U32) Software can no longer use this encoding as PrimitiveID is passed down the FF pipeline - see explanation above.

AddrMode

AddrMode

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	1

Addressing Mode This field determines the addressing method of the operand. Normally the destination operand and each source operand each have a distinct addressing mode field. When it is cleared, the register address of the operand is directly provided by bits in the instruction word. It is called a direct register addressing mode. When it is set, the register address of the operand is computed based on the address register value and an address immediate field in the instruction word. This is referred to as a register-indirect register addressing mode. This field applies to the destination operand and the first source operand, src0. Support for src1 is device dependent. See Table XX (Indirect source addressing support available in device hardware) in ISA Execution Environment for details.

Programming Notes

Instructions with 3 source operands use Direct Addressing.

Value	Name	Description
0	Direct	'Direct' register addressing
1	Indirect	'Register-Indirect' (or in short 'Indirect'). Register-indirect register addressing

ChanEn

ChanEn

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	1

Description

Channel Enables Dst.ChanEn, used only for the destination operand and only in the Align16 Access Mode. Provides four channel enable bits applied modulo four to all ExecSize channels. For example, 0xF enables all channels, 0 disables all channels, 0xA enables odd-numbered channels, and so on. The assembler mnemonics are x, y, z, and w for channels 0, 1, 2, and 3 respectively. If MaskCtrl is 1 (mnemonic NoMask) then all channels are enabled regardless of the ChanEn value, equivalent to ChanEn of 0xF (xyzw). Predication and execution masking, in addition to ChanEn and MaskCtrl, determine what channels are actually written.

Value	Name
0	Write Disabled
1	Write Enabled **[Default]**

ChanSel

ChanSel	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	2

Channel Select This field controls the channel swizzle for a source operand. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3.2], [5.4] and [7,6] are for channel 0 (?x?), 1 (?y?), 2 (?z?), and 3 (?w?) in the group, respectively. For example with an execution size of 8, r0.0<4>.zywz:f would assign the channels as follows: Chan0 = Data2, Chan1 = Data1, Chan2 = Data3, Chan3 = Data2; Chan4 = Data6, Chan5 = Data5, Chan6 = Data7, Chan7 = Data6. This field only applies to source operand. This field is only present in Align16 mode. It is not present for an immediate source operand. The 2-bit Channel Selection field for each channel within the group of 4 is defined as the following.

Programming Notes	Project
Note: When using channel select for 64-bit operands, the valid selects are .xy and .zw. This is required to pick a pair of DWords.	CHV, BSW

Value	Name	Description
00b	x	Channel 0 is selected for the corresponding execution channel
01b	y	Channel 1 is selected for the corresponding execution channel
10b	z	Channel 2 is selected for the corresponding execution channel
11b	w	Channel 3 is selected for the corresponding execution channel

Clear Color

Clear Color	
Project:	CHV, BSW
Source:	PRM
Size (in bits):	1

Value	Name	Description
0	CC_ZERO	Clear color value is 0.0, correctly interpreted based on surface format.
1	CC_ONE	Clear color value is 1.0, correctly interpreted based on surface format.

CondModifier

CondModifier	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	4

Conditional Modifier This field sets the flag register based on the internal conditional signals output from the execution pipe such as sign, zero, overflow and NaNs, etc. If this field is set to 0000, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field may also be referred to as the flag destination control field. This field applies to all instructions except send, sendc, and math.

Value	Name	Description
0000b	None **[Default]**	Do Not modify Flag Register
0001b	.z	Zero
0001b	.e	Equal
0010b	.nz	NotZero
0010b	.ne	NotEqual
0011b	.g	Greater-than
0100b	.ge	Greater-than-or-equal
0101b	.l	Less-than
0110b	.le	Less-than-or-equal
0111b	Reserved	
1000b	.o	Overflow
1001b	.u	Unordered with Computed NaN
1110b-1111b	Reserved	

DepCtrl

DepCtrl	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	2

Destination Dependency Control This field selectively disables destination dependency check and clear for this instruction. When it is set to 00, normal destination dependency control is performed for the instruction hardware checks for destination hazards to ensure data integrity. Specifically, destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard will be cleared when the destination operands retire. When bit 10 is set (NoDDClr), the destination register scoreboard will NOT be cleared when the destination operands retire. When bit 11 is set (NoDDChk), hardware does not check for destination register dependency before the instruction is made ready for execution. NoDDClr and NoDDChk are not mutual exclusive. When this field is not all-zero, hardware does not protect against destination hazards for the instruction. This is typically used to assemble data in a fine grained fashion (e.g. matrix-vector compute with dot-product instructions), where the data integrity is guaranteed by software based on the intended usage of instruction sequences.

Value	Name	Description
00b	None **[Default]**	Destination dependency checked and cleared (normal)
01b	NoDDClr	Destination dependency checked but not cleared
10b	NoDDChk	Destination dependency not checked but cleared
11b	NoDDClr, NoDDChk	Destination dependency not checked and not cleared

DstType

DstType	
Project:	CHV, BSW
Source:	EuIsa
Size (in bits):	4

Destination Type Numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send or sendc instruction, this field applies to CurrDst, the current destination operand. Three source instructions use a 3-bit encoding that allows fewer data types.

Value	Name	Description
0000b	:ud	Unsigned Doubleword integer
0001b	:d	signed Doubleword integer
0010b	:uw	Unsigned Word integer
0011b	:w	signed Word integer
0100b	:ub	Unsigned Byte integer
0101b	:b	signed Byte integer
0110b	:df	Double precision Float (64-bit)
0111b	:f	single precision Float (32-bit)
1000b	:uq	Unsigned Quadword integer
1001b	:q	signed Quadword integer
1010b	:hf	Half Float (16-bit)
1011b-1111b	Reserved	

EU_OPCODE

EU_OPCODE

Project: CHV, BSW
Source: Eulsa
Size (in bits): 7

Value	Name
40h	add
4Eh	addc
5h	and
0Ch	asr
42h	avg
18h	bfe
19h	bfi1
1Ah	bfi2
17h	bfrev
23h	brc
21h	brd
28h	break
2Ch	call
2Bh	calla
4Dh	cbit
10h	cmp
11h	cmpn
29h	cont
12h	csel
57h	dp2
56h	dp3
54h	dp4
55h	dph
24h	else
25h	endif
4Bh	fbh
4Ch	fbl
43h	frc
2Eh	goto

EU_OPCODE

2Ah	halt
22h	if
0h	illegal
20h	jmpi
2Fh	join
59h	line
5Ch	lrp
4Ah	lzd
48h	mac
49h	mach
5Bh	mad
5Dh	madm
38h	math
1h	mov
3h	movi
41h	mul
7Eh	nop
4h	not
6h	or
5Ah	pln
2Dh	ret
45h	rndd
46h	rnde
44h	rndu
47h	rndz
50h	sad2
51h	sada2
2h	sel
31h	send
32h	sendc
33h	sends
34h	sendsc
9h	shl
8h	shr
0Ah	smov
4Fh	subb

EU_OPCODE	
30h	wait
27h	while
7h	xor

ExecSize

ExecSize	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	3

Execution Size This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.

Restriction
An operand's Width must be less-than-or-equal to ExecSize

Value	Name	Programming Notes
000b	1 Channel (Scalar operation) **[Default]**	
001b	2 Channels	
010b	4 Channels	
011b	8 Channels	
100b	16 Channels	4-byte or smaller data types. Excludes DF, Q, and UQ types.
101b	32 Channels	2-byte or 1-byte data types. Excludes D, DF, F, Q, UD, and UQ types.
110b-111b	Reserved	

FC

FC	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	4

Math Function Control

Value	Name	Description
0000b	Reserved	
0001b	INV (reciprocal)	
0010b	LOG	
0011b	EXP	
0100b	SQRT	
0101b	RSQ	
0110b	SIN	
0111b	COS	
1000b	Reserved	
1001b	FDIV	
1010b	POW	
1011b	INT DIV BOTH	Return Quotient and Remainder
1100b	INT DIV QUOTIENT	Return Quotient Only
1101b	INT DIV REMAINDER	Return Remainder
1110b	INVM	
1111b	RSQRTM	

HorzStride

HorzStride	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	2

Description
Horizontal Stride This field provides the distance in unit of data elements between two adjacent data elements within a row (horizontal) in the register region for the operand. This field applies to both destination and source operands. This field is not present for an immediate source operand.
A horizontal stride of 0 is used for a row that is one-element wide, useful when an instruction repeats a column value or repeats a scalar value. For example, adding a single column to every column in a 2D array or adding a scalar to every element in a 2D array uses HorzStride of 0. A horizontal stride of 1 indicates that elements are adjacent within a row. References to HorzStride in this volume normally reference the value not the encoding, so there are references to HorzStride of 4, which is encoded as 11b.

Value	Name
00b	0 elements
01b	1 elements
10b	2 elements
11b	4 elements

PredCtrl

PredCtrl

Project: CHV, BSW
Source: Eulsa
Size (in bits): 4

Value	Name	Project	Exists If
0000b	No Predication (normal) **[Default]**		
0001b	Sequential Flag Channel Mapping		
0010b	Replication swizzle .x		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0010b	.anyv (any from f0.0-f0.1 on the same channel)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0011b	Replication swizzle .y		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0011b	.allv (all of f0.0-f0.1 on the same channel)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0100b	Replication swizzle .z		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0100b	.any2h (any in group of 2 channels)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0101b	Replication swizzle .w		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
0101b	.all2h (all in group of 2 channels)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
0110b	.any4h		
0111b	.all4h		
1000b-1111b	Reserved		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16')
1000b	.any8h (any in group of 8 channels)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1001b	.all8h (all in group of 8 channels)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1010b	.any16h (any in		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')

PredCtrl			
	group of 16 channels)		
1011b	.all16h (all in group of 16 channels)		(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1100b	.any32h (any in group of 32 channels)	CHV, BSW	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1101b	.all32h (all in group of 32 channels)	CHV, BSW	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')
1110b-1111b	Reserved	CHV, BSW	(Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1')

QtrCtrl

QtrCtrl

Project: CHV, BSW

Source: Eulsa

Size (in bits): 2

Quarter Control This field provides explicit control for ARF selection. This field combined with ExecSize determines which channels are used for the ARF registers. Along with NibCtrl, 1/8 DMask/VMask and ARF can be selected.

Programming Notes

NibCtrl is only allowed for SIMD4 instructions with a DF (Double Float) source or destination type.

Value	Name	Description	Programming Notes	Exists If
00b	1Q **[Default]**	Use first quarter for DMask/VMask. Use first half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
01b	2Q	Use second quarter for DMask/VMask. Use second half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
10b	3Q	Use third quarter for DMask/VMask. Use first half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
11b	4Q	Use fourth quarter for DMask/VMask. Use second half for everything else.		([ExecSize]=='8') AND ([NibCtrl]=='0')
0	1H	Use first half for DMask/VMask. Use all channels for everything else.		([ExecSize]=='16') AND ([NibCtrl]=='0')
2	2H	Use second half for DMask/VMask. Use all channels for everything else.	Only allowed for SIMD16 instruction in Single Program Flow mode (SPF=1)	([ExecSize]=='16') AND ([NibCtrl]=='0')
0	1N	Use first 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
0	2N	Use second 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
1	3N	Use third 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
1	4N	Use fourth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
2	5N	Use fifth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')

QtrCtrl				
2	6N	Use sixth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')
3	7N	Use seventh 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='0')
3	8N	Use eighth 1/8th for DMask/VMask and ARF.		([ExecSize]=='4') AND ([NibCtrl]=='1')

RegFile

RegFile

Project: CHV, BSW

Source: Eulsa

Size (in bits): 2

Value	Name	Description	Programming Notes
00b	ARF	Architecture Register File	Only allowed for src0 or destination
01b	GRF	General Register File - allowed for any source or destination	
10b	Reserved		
11b	IMM	Immediate operand	Only allowed for the last source operand. Not allowed for the destination operand or for any other source operand. Note that for flow control instructions requiring two offsets, regfile of source0 is required to be immediate since the 64b for immediates occupy the DW2 and DW3

RepCtrl

RepCtrl	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	1

Replicate Control. This field is only present in three-source instructions, for each of the three source operands. It controls replication of the starting channel to all channels in the execution size. This is applicable to 32b datatypes. 16b and 64b datatypes cannot use the replicate control.

Value	Name
0	No replication
1	Replicate across all channels

SFID

SFID

Project: CHV, BSW

Source: Eulsa

Size (in bits): 4

The following table lists the assignments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a 'send' instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function. Note: The initial intention was to combine these two ID namespaces, so that (theoretically) an agent (such as the Thread Spawner) that served both as a Shared Function and Fixed Function would have a single, unique 4-bit ID encoding. However, this combination is not a requirement of the architecture.

Programming Notes

SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.

Value	Name	Description	Programming Notes
0000b	SFID_NULL	Null	
0001b	Reserved	Reserved	
0010b	SFID_SAMPLER	Sampler	
0011b	SFID_GATEWAY	Message Gateway	
0100b	SFID_DP_SAMPLER	Sampler Cache Data Port	
0101b	SFID_DP_RC	Render Cache Data Port	
0110b	SFID_URB	URB	
0111b	SFID_SPAWNER	Thread Spawner	
1000b	SFID_VME	Video Motion Estimation	
1001b	SFID_DP_CC	Constant Cache Data Port	
1010b	SFID_DP_DC0	Data Cache Data Port	
1011b	SFID_PI	Pixel Interpolator	
1100b	SFID_DP_DC1	Data Cache Data Port 1	SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity.
1101b	SFID_CRE	Check and Refinement Engine	
1110b-1111b	Reserved		

Shader Channel Select

Shader Channel Select

Project: CHV, BSW
Source: PRM
Size (in bits): 3

Value	Name	Description
0	ZERO	
1	ONE	
2	Reserved	
3	Reserved	
4	RED	Shader channel is set to surface red channel
5	GREEN	Shader channel is set to surface green channel
6	BLUE	Shader channel is set to surface blue channel
7	ALPHA	Shader channel is set to surface alpha channel

SIMD Mode

SIMD Mode

Project: All
Source: PRM
Size (in bits): 2

Value	Name
0	SIMD4x2
1	SIMD8
2	SIMD16
3	SIMD32/64

SrcImmType

SrcImmType	
Project:	CHV, BSW
Source:	EuIsa
Size (in bits):	4

Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF. Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions. In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW. Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding. The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands.

Value	Name	Description
0000b	:ud	Unsigned Doubleword
0001b	:d	signed Doubleword
0010b	:uw	Unsigned Word integer
0011b	:w	signed Word integer
0100b	:uv	Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer.
0101b	:vf	Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number
0110b	:v	Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer
0111b	:f	single precision Float (32-bit)
1000b	:uq	Unsigned Quadword integer
1001b	:q	signed Quadword integer
1010b	:df	Double precision Float (64-bit)
1011b	:hf	Half Float (16-bit)
1100b-1111b	Reserved	

SrcIndex

SrcIndex

Project: CHV, BSW
Source: Eulsa
Size (in bits): 5

Value	Name	Description
0	000000000000	dir | <0;1,0>
1	000000000010	(-) | dir | <0;1,0>
2	000000010000	dir | <0;>.zx
3	000000010010	(-) | dir | <0;>.zx
4	000000011000	dir | <0;>.wx
5	000000100000	dir | <0;>.xy
6	000000101000	dir | <0;>.yy
7	000001001000	dir | <0;4,1>
8	000001010000	dir | <0;>.zz
9	000001110000	dir | <0;>.zw
10	000001111000	dir | <0;8,4> / dir | <0;>.ww
11	001100000000	dir | <4;>.xx
12	001100000010	(-) | dir | <4;>.xx
13	001100001000	dir | <4;>.yx
14	001100010000	dir | <4;>.zx
15	001100010010	(-) | dir | <4;>.zx
16	001100100000	dir | <4;>.xy
17	001100101000	dir | <4;>.yy
18	001100111000	dir | <4;>.wy
19	001101000000	dir | <4;4,0>
20	001101000010	(-) | dir | <4;4,0>
21	001101001000	dir | <4;>.yz
22	001101010000	dir | <4;>.zz
23	001101100000	dir | <4;>.xw
24	001101101000	dir | <4;>.yw
25	001101110000	dir | <4;>.zw
26	001101110001	(abs) | dir | <4;>.zw
27	001101111000	dir | <4;>.ww
28	010001101000	dir | <8;8,1>

SrcIndex		
29	010001101001	(abs) | dir | <8;8,1>
30	010001101010	(-) | dir | <8;8,1>
31	010110001000	dir | <16;16,1>

SrcMod

SrcMod

Project: CHV, BSW

Source: Eulsa

Size (in bits): 2

Description

Source Modifier This field specifies the numeric modification of a source operand. The value of each data element of a source operand can optionally have its absolute value taken and/or its sign inverted prior to delivery to the execution pipe. The absolute value is prior to negate such that a guaranteed negative value can be produced. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

When used with logic instructions (and, not, or, xor), this field indicates whether the source bits are inverted (bitwise NOT) before delivery to the execution pipe, regardless of the source type.

Value	Name	Description
00b	No modification	
01b	abs	Absolute value Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)
10b	negate	Negate Logic instructions: Bitwise NOT, inverting the source bits
11b	negate of abs	Negate of the absolute (forced negative value) Logic instructions: No modification (This encoding cannot be selected in the assembler syntax)

SrcType

SrcType	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	4

Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF. Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions. In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW. Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding. The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands.

Value	Name	Description
0000b	:ud	Unsigned Doubleword
0001b	:d	signed Doubleword
0010b	:uw	Unsigned Word integer
0011b	:w	signed Word integer
0100b	:ub	unsigned Byte integer
0101b	:b	signed Byte integer
0110b	:df	Double precision Float (64-bit)
0111b	:f	single precision Float (32-bit)
1000b	:uq	Unsigned Quadword integer
1001b	:q	signed Quadword integer
1010b	:hf	Half Float (16-bit)
1011b-1111b	Reserved	

SURFACE_FORMAT

SURFACE_FORMAT

Project: All

Source: PRM

Size (in bits): 9

The following table indicates the supported surface formats and the 9-bit encoding for each. Note that some of these formats are used not only by the Sampling Engine, but also by the Data Port and the Vertex Fetch unit.

Value	Name	Bits Per Element (BPE)	Description
000h	R32G32B32A32_FLOAT	128	
001h	R32G32B32A32_SINT	128	
002h	R32G32B32A32_UINT	128	
003h	R32G32B32A32_UNORM	128	
004h	R32G32B32A32_SNORM	128	
005h	R64G64_FLOAT	128	
006h	R32G32B32X32_FLOAT	128	
007h	R32G32B32A32_SSCALED	128	
008h	R32G32B32A32_USCALED	128	
020h	R32G32B32A32_SFIXED	128	
021h	R64G64_PASSTHRU	128	
040h	R32G32B32_FLOAT	96	
041h	R32G32B32_SINT	96	
042h	R32G32B32_UINT	96	
043h	R32G32B32_UNORM	96	
044h	R32G32B32_SNORM	96	
045h	R32G32B32_SSCALED	96	
046h	R32G32B32_USCALED	96	
050h	R32G32B32_SFIXED	96	
080h	R16G16B16A16_UNORM	64	
081h	R16G16B16A16_SNORM	64	
082h	R16G16B16A16_SINT	64	
083h	R16G16B16A16_UINT	64	
084h	R16G16B16A16_FLOAT	64	
085h	R32G32_FLOAT	64	
086h	R32G32_SINT	64	
087h	R32G32_UINT	64	

SURFACE_FORMAT

088h	R32_FLOAT_X8X24_TYPELESS	64	
089h	X32_TYPELESS_G8X24_UINT	64	
08Ah	L32A32_FLOAT	64	
08Bh	R32G32_UNORM	64	
08Ch	R32G32_SNORM	64	
08Dh	R64_FLOAT	64	
08Eh	R16G16B16X16_UNORM	64	
08Fh	R16G16B16X16_FLOAT	64	
090h	A32X32_FLOAT	64	
091h	L32X32_FLOAT	64	
092h	I32X32_FLOAT	64	
093h	R16G16B16A16_SSCALED	64	
094h	R16G16B16A16_USCALED	64	
095h	R32G32_SSCALED	64	
096h	R32G32_USCALED	64	
0A0h	R32G32_SFIXED	64	
0A1h	R64_PASSTHRU	64	
0C0h	B8G8R8A8_UNORM	32	
0C1h	B8G8R8A8_UNORM_SRGB	32	
0C2h	R10G10B10A2_UNORM	32	
0C3h	R10G10B10A2_UNORM_SRGB	32	
0C4h	R10G10B10A2_UINT	32	
0C5h	R10G10B10_SNORM_A2_UNORM	32	
0C7h	R8G8B8A8_UNORM	32	
0C8h	R8G8B8A8_UNORM_SRGB	32	
0C9h	R8G8B8A8_SNORM	32	
0CAh	R8G8B8A8_SINT	32	
0CBh	R8G8B8A8_UINT	32	
0CCh	R16G16_UNORM	32	
0CDh	R16G16_SNORM	32	
0CEh	R16G16_SINT	32	
0CFh	R16G16_UINT	32	
0D0h	R16G16_FLOAT	32	
0D1h	B10G10R10A2_UNORM	32	
0D2h	B10G10R10A2_UNORM_SRGB	32	
0D3h	R11G11B10_FLOAT	32	

SURFACE_FORMAT

0D6h	R32_SINT	32	
0D7h	R32_UINT	32	
0D8h	R32_FLOAT	32	
0D9h	R24_UNORM_X8_TYPELESS	32	
0DAh	X24_TYPELESS_G8_UINT	32	
0DDh	L32_UNORM	32	
0DEh	A32_UNORM	32	
0DFh	L16A16_UNORM	32	
0E0h	I24X8_UNORM	32	
0E1h	L24X8_UNORM	32	
0E2h	A24X8_UNORM	32	
0E3h	I32_FLOAT	32	
0E4h	L32_FLOAT	32	
0E5h	A32_FLOAT	32	
0E6h	X8B8_UNORM_G8R8_SNORM	32	
0E7h	A8X8_UNORM_G8R8_SNORM	32	
0E8h	B8X8_UNORM_G8R8_SNORM	32	
0E9h	B8G8R8X8_UNORM	32	
0EAh	B8G8R8X8_UNORM_SRGB	32	
0EBh	R8G8B8X8_UNORM	32	
0ECh	R8G8B8X8_UNORM_SRGB	32	
0EDh	R9G9B9E5_SHAREDEXP	32	
0EEh	B10G10R10X2_UNORM	32	
0F0h	L16A16_FLOAT	32	
0F1h	R32_UNORM	32	
0F2h	R32_SNORM	32	
0F3h	R10G10B10X2_USCALED	32	
0F4h	R8G8B8A8_SSCALED	32	
0F5h	R8G8B8A8_USCALED	32	
0F6h	R16G16_SSCALED	32	
0F7h	R16G16_USCALED	32	
0F8h	R32_SSCALED	32	
0F9h	R32_USCALED	32	
100h	B5G6R5_UNORM	16	
101h	B5G6R5_UNORM_SRGB	16	
102h	B5G5R5A1_UNORM	16	

SURFACE_FORMAT			
103h	B5G5R5A1_UNORM_SRGB	16	
104h	B4G4R4A4_UNORM	16	
105h	B4G4R4A4_UNORM_SRGB	16	
106h	R8G8_UNORM	16	
107h	R8G8_SNORM	16	
108h	R8G8_SINT	16	
109h	R8G8_UINT	16	
10Ah	R16_UNORM	16	
10Bh	R16_SNORM	16	
10Ch	R16_SINT	16	
10Dh	R16_UINT	16	
10Eh	R16_FLOAT	16	
10Fh	A8P8_UNORM_PALETTE0	16	
110h	A8P8_UNORM_PALETTE1	16	
111h	I16_UNORM	16	
112h	L16_UNORM	16	
113h	A16_UNORM	16	
114h	L8A8_UNORM	16	
115h	I16_FLOAT	16	
116h	L16_FLOAT	16	
117h	A16_FLOAT	16	
118h	L8A8_UNORM_SRGB	16	
119h	R5G5_SNORM_B6_UNORM	16	
11Ah	B5G5R5X1_UNORM	16	
11Bh	B5G5R5X1_UNORM_SRGB	16	
11Ch	R8G8_SSCALED	16	
11Dh	R8G8_USCALED	16	
11Eh	R16_SSCALED	16	
11Fh	R16_USCALED	16	
122h	P8A8_UNORM_PALETTE0	16	
123h	P8A8_UNORM_PALETTE1	16	
124h	A1B5G5R5_UNORM	16	
125h	A4B4G4R4_UNORM	16	
126h	L8A8_UINT	16	
127h	L8A8_SINT	16	
140h	R8_UNORM	8	

SURFACE_FORMAT

141h	R8_SNORM	8	
142h	R8_SINT	8	
143h	R8_UINT	8	
144h	A8_UNORM	8	
145h	I8_UNORM	8	
146h	L8_UNORM	8	
147h	P4A4_UNORM_PALETTE0	8	
148h	A4P4_UNORM_PALETTE0	8	
149h	R8_SSCALED	8	
14Ah	R8_USCALED	8	
14Bh	P8_UNORM_PALETTE0	8	
14Ch	L8_UNORM_SRGB	8	
14Dh	P8_UNORM_PALETTE1	8	
14Eh	P4A4_UNORM_PALETTE1	8	
14Fh	A4P4_UNORM_PALETTE1	8	
150h	Y8_UNORM	8	
152h	L8_UINT	8	
153h	L8_SINT	8	
154h	I8_UINT	8	
155h	I8_SINT	8	
180h	DXT1_RGB_SRGB	0	
181h	R1_UNORM	1	
182h	YCRCB_NORMAL	0	
183h	YCRCB_SWAPUVY	0	
184h	P2_UNORM_PALETTE0	2	
185h	P2_UNORM_PALETTE1	2	
186h	BC1_UNORM	0	(DXT1)
187h	BC2_UNORM	0	(DXT2/3)
188h	BC3_UNORM	0	(DXT4/5)
189h	BC4_UNORM	0	
18Ah	BC5_UNORM	0	
18Bh	BC1_UNORM_SRGB	0	(DXT1_SRGB)
18Ch	BC2_UNORM_SRGB	0	(DXT2/3_SRGB)
18Dh	BC3_UNORM_SRGB	0	(DXT4/5_SRGB)
18Eh	MONO8	1	
18Fh	YCRCB_SWAPUV	0	

SURFACE_FORMAT

190h	YCRCB_SWAPY	0	
191h	DXT1_RGB	0	
192h	FXT1	0	
193h	R8G8B8_UNORM	24	
194h	R8G8B8_SNORM	24	
195h	R8G8B8_SSCALED	24	
196h	R8G8B8_USCALED	24	
197h	R64G64B64A64_FLOAT	256	
198h	R64G64B64_FLOAT	192	
199h	BC4_SNORM	0	
19Ah	BC5_SNORM	0	
19Bh	R16G16B16_FLOAT	48	
19Ch	R16G16B16_UNORM	48	
19Dh	R16G16B16_SNORM	48	
19Eh	R16G16B16_SSCALED	48	
19Fh	R16G16B16_USCALED	48	
1A1h	BC6H_SF16	0	
1A2h	BC7_UNORM	0	
1A3h	BC7_UNORM_SRGB	0	
1A4h	BC6H_UF16	0	
1A5h	PLANAR_420_8	0	
1A8h	R8G8B8_UNORM_SRGB	24	
1A9h	ETC1_RGB8	0	
1AAh	ETC2_RGB8	0	
1ABh	EAC_R11	0	
1ACh	EAC_RG11	0	
1ADh	EAC_SIGNED_R11	0	
1AEh	EAC_SIGNED_RG11	0	
1AFh	ETC2_SRGB8	0	
1B0h	R16G16B16_UINT	48	
1B1h	R16G16B16_SINT	48	
1B2h	R32_SFIXED	32	
1B3h	R10G10B10A2_SNORM	32	
1B4h	R10G10B10A2_USCALED	32	
1B5h	R10G10B10A2_SSCALED	32	
1B6h	R10G10B10A2_SINT	32	

SURFACE_FORMAT

1B7h	B10G10R10A2_SNORM	32	
1B8h	B10G10R10A2_USCALED	32	
1B9h	B10G10R10A2_SSCALED	32	
1BAh	B10G10R10A2_UINT	32	
1BBh	B10G10R10A2_SINT	32	
1BCh	R64G64B64A64_PASSTHRU	256	
1BDh	R64G64B64_PASSTHRU	192	
1C0h	ETC2_RGB8_PTA	0	
1C1h	ETC2_SRGB8_PTA	0	
1C2h	ETC2_EAC_RGBA8	0	
1C3h	ETC2_EAC_SRGB8_A8	0	
1C8h	R8G8B8_UINT	24	
1C9h	R8G8B8_SINT	24	
1FFh	RAW	0	

Texture Coordinate Mode

Texture Coordinate Mode

Project: CHV, BSW
Source: PRM
Size (in bits): 3

Value	Name	Description
0h	WRAP	Map is repeated in the U direction
1h	MIRROR	Map is mirrored in the U direction
2h	CLAMP	Map is clamped to the edges of the accessed map
3h	CUBE	For cube-mapping, filtering in edges access adjacent map faces
4h	CLAMP_BORDER	Map is infinitely extended with the border color
5h	MIRROR_ONCE	Map is mirrored once about origin, then clamped
6h	HALF_BORDER	Map is infinitely extended with the average of the nearest edge texel and the border color
7h	Reserved	

ThreadCtrl

ThreadCtrl	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	2

Thread Control This field provides explicit control for thread switching.

Value	Name	Description
00b	Normal	Up to the GEN execution units to manage thread switching. This is the normal (and unnamed) mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. In another example, if the current instruction is ready to go, however, there is another thread with higher priority that also has an instruction ready, the EU switches to that thread. Execution may or may not be preempted by another thread following this instruction.
01b	Atomic	Prevent any thread switch immediately following this instruction. Always execute the next instruction (which may not be next sequentially if the current instruction branches). The next instruction gets highest priority in the thread arbitration for the execution pipelines.
10b	Switch	A forced thread switch occurs after the current instruction is executed and before the next instruction. In addition, a long delay (longer than the execution pipe latency) is introduced for the current thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. Switch is designed primarily as a safety feature in case there are race conditions for certain instructions. Force a switch to another thread after this instruction and before the next instruction.
11b	Reserved	

VertStride

VertStride	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	4

Vertical Stride The field provides the vertical stride of the register region in unit of data elements for an operand. Encoding of this field provides values of 0 or powers of 2, ranging from 1 to 32 elements. Larger values are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). Special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (AddrMode = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute the addresses. Each address sub-register provides the origin for a row of data element. The number of address sub-registers used is determined by the division of ExecSize of the instruction by the Width fields of the operand. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

Programming Notes
Note 1: Vertical Stride larger than 32 is not allowed due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total).
Note 2: In Align16 access mode, as encoding 0xF is reserved, only single-index indirect addressing is supported.
Note 3: If indirect address is supported for src1, encoding 0xF is reserved for src1 only single-index indirect addressing is supported.
Note 4: Encoding 0010 applies for QWord-size operands.

Value	Name	Programming Notes
0000b	0 elements	
0001b	1 element	Align1 mode only.
0010b	2 elements	
0011b	4 elements	
0100b	8 elements	Align1 mode only.
0101b	16 elements	Applies to byte or word operand only. Align1 mode only.
0110b	32 elements	Applies to byte operand only. Align1 mode only.
0111b-1110b	Reserved	
1111b	VxH or Vx1 mode	Only valid for register-indirect addressing in Align1 mode.

Width

Width

Project: CHV, BSW

Source: Eulsa

Size (in bits): 3

This field specifies the number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the ExecSize field of the instruction. This field only applies to source operand. It does not apply to destination. This field is not present for an immediate source operand.

Programming Notes

Note that with ExecSize of 32, because the maximum Width is 16, there are at least two rows in a source region.

Value	Name
000b	1 elements
001b	2 elements
010b	4 elements
011b	8 elements
100b	16 elements
101b-111b	Reserved

WRAP_SHORTEST_ENABLE

WRAP_SHORTEST_ENABLE	
Source:	RenderCS
Size (in bits):	4

This state selects which components (if any) of Attribute [n] are to be interpolated in a "wrap shortest" fashion. Operation is UNDEFINED if any of these bits are set and the Constant Interpolation Enable bit associated with this attribute is set. Note that wrap-shortest interpolation is only supported for Attributes 0-15.

Value	Name	Description
0001b	X	Wrap Shortest X Component
0010b	Y	Wrap Shortest Y Component
0011b	XY	Wrap Shortest XY Components
0100b	Z	Wrap Shortest Z Component
0101b	XZ	Wrap Shortest XZ Components
0110b	YZ	Wrap Shortest YZ Components
0111b	XYZ	Wrap Shortest XYZ Components
1000b	W	Wrap Shortest W Component
1001b	XW	Wrap Shortest XW Components
1010b	YW	Wrap Shortest YW Components
1011b	XYW	Wrap Shortest XYW Components
1100b	ZW	Wrap Shortest ZW Components
1101b	XZW	Wrap Shortest XZW Components
1110b	YZW	Wrap Shortest YZW Components
1111b	XYZW	Wrap Shortest XYZW Components

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 2c: Command Reference: Register Addresses

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Advanced Features Length and Capabilities 1
Advanced Scheduler Reset Request Messages 3
Aggregate_Perf_Counter_A31 5
Aggregate_Perf_Counter_A32 6
Aggregate_Perf_Counter_A33 7
Aggregate_Perf_Counter_A34 8
Aggregate_Perf_Counter_A35 9
Aggregate Perf Counter A1 10
Aggregate Perf Counter A5 11
Aggregate Perf Counter A7 12
Aggregate Perf Counter A8 13
Aggregate Perf Counter A9 14
Aggregate Perf Counter A10 15
Aggregate Perf Counter A11 16
Aggregate Perf Counter A12 17
Aggregate Perf Counter A13 18
Aggregate Perf Counter A14 19
Aggregate Perf Counter A15 20
Aggregate Perf Counter A16 21
Aggregate Perf Counter A17 22
Aggregate Perf Counter A18 23
Aggregate Perf Counter A21 24
Aggregate Perf Counter A22 25
Aggregate Perf Counter A23 26
Aggregate Perf Counter A24 27
Aggregate Perf Counter A25 28
Aggregate Perf Counter A26 29
Aggregate Perf Counter A27 30
Aggregate Perf Counter A28 31
Aggregate Perf Counter A29 32
Aggregate Perf Counter A30 33
AFCTLSTS 34

All Engine Fault Register 35
ARAT CUTRIG HI 36
ARAT CUTRIG LO 37
ARAT Delta (LSB) 38
ARAT Delta (MSB) 39
ARAT POST CU BUSY 40
ARAT PRE CU BUSY 41
Arbiter Control Register 42
Arbiter Mode Control Register 44
ASLS 47
ASYNC_SLICE_COUNT 48
Auto Draw End Offset 49
Base of Data Stolen Memory 50
Base of GTT Stolen Memory 51
Batch Address Difference Register 52
Batch Buffer Head Pointer Register 53
Batch Buffer Per Context Pointer 55
Batch Buffer Start Head Pointer Register 58
Batch Buffer Start Head Pointer Register for Upper DWord 60
Batch Buffer State Register 61
Batch Buffer Upper Head Pointer Preemption Register 64
Batch Buffer Upper Head Pointer Register 65
BCS_PREEMPTION_HINT 66
BCS_PREEMPTION_HINT_UDW 68
BCS Active Upper Head Pointer Register 69
BCS Context ID Preemption Hint 70
BCS Context Sizes 71
BCS Context Timestamp Count 72
BCS Counter for the Blitter Engine 73
BCS Error Identity Register 74
BCS Error Mask Register 75
BCS Error Status Register 76
BCS Execute Condition Code Register 77

BCS General Purpose Register 79
BCS Hardware Status Mask Register 80
BCS IDLE Max Count 81
BCS Idle Switch Delay 82
BCS Instruction Parser Mode Register 83
BCS Interrupt Mask Register 84
BCS Mode Register for Software Interface 85
BCS PPGTT Directory Cacheline Valid Register 87
BCS Primary DMA Engine Fetch Upper Address 88
BCS Reported Timestamp Count 89
BCS Reset Control Register 90
BCS Ring Buffer Next Context ID Register 91
BCS Semaphore Polling Interval on Wait 92
BCS SW Control 93
BCS Watchdog Counter Threshold 94
Bitstream Output Bit Count for the last Syntax Element Report Register 95
Bitstream Output Byte Count Per Slice Report Register 96
Bitstream Output Minimal Size Padding Count Report Register 97
Blitter Mode Register 98
Blitter TLB Control Register 101
BLT Context Element Descriptor (High Part) 102
BLT Context Element Descriptor (Low Part) 103
BLT Context Element Descriptor (Low Part) 104
BLT PDP0/PML4/PASID Descriptor (High Part) 105
BLT PDP0/PML4/PASID Descriptor (Low Part) 106
BLT PDP1 Descriptor Register (High Part) 107
BLT PDP1 Descriptor Register (Low Part) 108
BLT PDP2 Descriptor Register (High Part) 109
BLT PDP2 Descriptor Register (Low Part) 110
BLT PDP3 Descriptor Register (High Part) 111
BLT PDP3 Descriptor Register (Low Part) 112
Boolean_Counter_B0 113
Boolean_Counter_B1 114

Boolean_Counter_B2 115
Boolean_Counter_B3 116
Boolean_Counter_B4 117
Boolean_Counter_B5 118
Boolean_Counter_B6 119
Boolean_Counter_B7 120
BTB Not Consumed By RCS 121
BTP Commands Parsed By RCS 122
Cache Mode Register 0 123
Cache Mode Register 1 127
CAPPOINT 133
CCK_CTL1 134
CCK_CTL2 137
CCK_CTL3 139
CCK_CTL4 140
CLAIM_ER 141
Clipper Invocation Counter 142
Clipper Primitives Counter 143
CLKGATE Messaging Register for Clocking Unit 144
CLOCK_GATE_DIS1 146
CLOCK_GATE_DIS2 150
Clock gate and clear EU metrics 152
Clock Gating Messages 154
Color/Depth Write FIFO Watermarks 156
Configuration Register0 for RPMunit 157
Configuration Register1 for RPMunit 158
Configuration Register for RCPunit 159
Context Load Protocol Register BLT 161
Context Load Protocol Register CS 164
Context Load Protocol Register VCS0 167
Context Load Protocol Register VCS1 170
Context Load Protocol Register VEBX 173
Context Restore Request To TDL 176

Context Save Request To TDL....177
Context Sizes....178
Context Status Buffer Contents....179
Control Register for Power Management....181
Count Active Channels Dispatched....185
CS Context Timestamp Count....186
CS General Purpose Register....187
CS Power Management FSM....190
CSPREEMPT....194
CS Reset Control Register....195
Current Context Register....196
Current Idle/Busy/Avg Count for Freq Down Recommendation....198
Current Idle/Busy/Avg Count for Freq Up Recommendation....199
Current Time in DOWN EI....200
Current Time in UP EI....201
Customizable Event Creation 0-0....202
Customizable Event Creation 1-0....204
Customizable Event Creation 2-0....206
Customizable Event Creation 3-0....208
Customizable Event Creation 4-0....210
Customizable Event Creation 5-0....212
Customizable Event Creation 6-0....214
Customizable Event Creation 7-0....216
CVS TLB LRA 0....218
CVS TLB LRA 1....220
CVS TLB LRA 2....222
Depth/Early Depth TLB Partitioning Register....223
DID....224
DS Invocation Counter....226
DX9 Constants Not Consumed By RCS....227
DX9 Constants Prsed By RCS....228
ECO Bits - Bus Reset Domain with lock bit....229
ECO Bits - Device Reset Domain....231

ECO Message Register....232
ECO Reserved....234
ECREQ....235
Element Descriptor Register....236
Error_Identity_Reg....239
Error_Mask_Reg....241
Error_Status_Reg....243
Error Identity Register....245
Error Mask Register....246
Error Reporting Register....247
Error Status Register....249
EU Mask Programming....250
EU Metrics for Event0 High....253
EU Metrics for Event0 Low....254
EU Metrics for Event1 High....255
EU Metrics for Event1 Low....256
EU Metrics for Event2 High....257
EU Metrics for Event2 Low....258
EU Metrics for Event3 High....259
EU Metrics for Event3 Low....260
EU Metrics for Event4 High....261
EU Metrics for Event4 Low....262
EU Metrics for Event5 High....263
EU Metrics for Event5 Low....264
EU Metrics for Event6 High....265
EU Metrics for Event6 Low....266
EU Metrics for Event7 High....267
EU Metrics for Event7 Low....268
EU Metrics for Event8 High....269
EU Metrics for Event8 Low....270
EU Metrics for Event9 High....271
EU Metrics for Event9 Low....272
EU Metrics for Event10 High....273

EU Metrics for Event10 Low 274
EU Metrics for Event11 High 275
EU Metrics for Event11 Low 276
EU Metrics for Event12 High 277
EU Metrics for Event12 Low 278
Event selection and base counters 279
Event Selection and Base Counters1 282
Execlist Status 283
Execlist Submit Port Register 286
Execute Condition Code Register 289
FAULT_TLB_RD_DATA0 Register 291
FAULT_TLB_RD_DATA1 Register 292
Fault Switch Out 293
FBC_RT_BASE_ADDR_REGISTER 294
FBC_RT_BASE_ADDR_REGISTER_UPPER 296
FD 297
FF Performance 298
First Buffer Size and Start 300
Flexible EU Event Control 0 302
Flexible EU Event Control 1 304
Flexible EU Event Control 2 306
Flexible EU Event Control 3 308
Flexible EU Event Control 4 310
Flexible EU Event Control 5 312
Flexible EU Event Control 6 314
FORCE_TO_NONPRIV 316
Frame count and Draw call number 322
FUSEWORD0 323
FUSEWORD1 324
FUSEWORD2 325
FUSEWORD3 326
FUSEWORD4 327
FUSEWORD5 328

FUSEWORD6 329
FUSEWORD7 330
FUSEWORD8 331
FUSEWORD9 332
FUSEWORD10 333
FUSEWORD11 334
FUSEWORD12 335
FUSEWORD13 336
FUSEWORD14 337
FUSEWORD15 338
FUSEWORD16 339
FUSEWORD17 340
FUSEWORD18 341
FUSEWORD19 342
G3D Control Register 343
GAB Arbitration Programmable 344
GAB GAC GAM Idle 345
GAB LRA 0 346
GAB LRA 1 348
GAB unit Control Register 349
GAC_GAM Arbitration Counters Register 0 350
GAC_GAM Arbitration Counters Register 1 351
GAC_GAM R Arbitration Register 0 352
GAC_GAM R Arbitration Register 1 353
GAC_GAM R Arbitration Register 2 354
GAC_GAM R Arbitration Register 3 355
GAC_GAM RO Arbitration Register 0 356
GAC_GAM RO Arbitration Register 1 357
GAC_GAM RO Arbitration Register 2 358
GAC_GAM RO Arbitration Register 3 359
GAC_GAM WR Arbitration Register 0 360
GAC_GAM WR Arbitration Register 1 361
GAC_GAM WR Arbitration Register 2 362

GAC_GAM WR Arbitration Register 3 363
GAM and SA Communication Register 364
Gam Fub Done1 Lookup Register 367
Gam Fub Done Lookup Register 368
GAM Put Delay 370
GAMT_DONE Register 371
GAMT_ECO_REG_RO_IA 373
GAMT_ECO_REG_RW_IA 374
GAMT Arbiter Mode Control 375
GAMW_ECO_BUS_RO_IA 378
GAMW_ECO_BUS_RW_IA 379
GAMW_ECO_DEV_RO_IA 380
GAMW_ECO_DEV_RW_IA 381
GAMW Power Context Save 382
GARB Messaging Register for Boot Controller 384
GARB Messaging Register for Clocking Unit 386
Gather Constants Not Consumed By RCS 387
GDR Per Client Write Drop Enables 388
GDR Write Drop 390
General Purpose Power Management Performance Idle Hysteresis 391
GFX_FLSH_CNT 392
GFX Arbiter Client Priority Control 393
GFX Context Element Descriptor (High Part) 395
GFX Context Element Descriptor (Low Part) 396
GFX Context Element Descriptor (Low Part) 398
GFX PDP0/PML4/PASID Descriptor (High Part) 400
GFX PDP0/PML4/PASID Descriptor (Low Part) 401
GFX PDP1 Descriptor Register (High Part) 402
GFX PDP1 Descriptor Register (Low Part) 403
GFX PDP2 Descriptor Register (High Part) 404
GFX PDP2 Descriptor Register (Low Part) 405
GFX PDP3 Descriptor Register (High Part) 406
GFX PDP3 Descriptor Register (Low Part) 407

GGC 408
Global Clear Value Register 0 411
Global Clear Value Register 1 412
Global Clear Value Register 2 413
Global Clear Value Register 3 414
Global Invalidation Register 415
Global System Interrupt Routine 416
GMADR_LSB 417
GMADR_MSB 419
GMBC Message Register 420
GO Messaging Register for GAMunit 422
Go Protocol GAM Request 424
GPA to HPA Translation Request 428
GPA value for GPA to HPA Translation 431
GPGPU Context Restore Request To TDL 432
GPGPU Context Save Request To TDL 434
GPGPU Dispatch Dimension X 435
GPGPU Dispatch Dimension Y 436
GPGPU Dispatch Dimension Z 437
GPU_Ticks_Counter 438
Graphics Device Reset Control 439
Graphics Mode Register 441
GS Domain Clock Gate Control Register 445
GS Invocation Counter 448
GS Primitives Counter 449
GT_CR_POWER_METER_CTRL 450
GTFIFOCTL 452
GT Function Level Reset Control Message 454
GT INTERRUPT 0 ENABLE REGISTER 455
GT INTERRUPT 0 IDENTITY REGISTER 458
GT INTERRUPT 0 MASK REGISTER 461
GT INTERRUPT 0 STATUS REGISTER 465
GT INTERRUPT1 ENABLE REGISTER 468

GT INTERRUPT1 IDENTITY REGISTER 471
GT INTERRUPT1 MASK REGISTER 474
GT INTERRUPT1 STATUS REGISTER 477
GT INTERRUPT3 ENABLE REGISTER 480
GT INTERRUPT3 IDENTITY REGISTER 482
GT INTERRUPT3 MASK REGISTER 484
GT INTERRUPT3 STATUS REGISTER 486
GTLC_PW_STAT 488
GTLC_SURVIVE 490
GTLC_WAKE 492
GT Mode Register 494
GTSCRATCH1 497
GTSCRATCH2 498
GTSCRATCH3 499
GTSCRATCH4 500
GTSCRATCH5 501
GTSCRATCH6 502
GTSCRATCH7 503
GTSCRATCH8 504
GTSCRATCHPAD0 505
GTSCRATCHPAD1 506
GTSCRATCHPAD2 507
GTSCRATCHPAD3 508
GTSCRATCHPAD4 509
GTSCRATCHPAD5 510
GTSCRATCHPAD6 511
GTSCRATCHPAD7 512
GTT Cache Enable 513
GTTMMADR_LSB 514
GTTMMADR_MSB 516
GU_CTL0 517
Hardware Status Mask Register 520
Hardware Status Page Address Register 522

HCP CABAC Status 524
HCP Decode Status 526
HCP Picture Checksum cIdx0 527
HCP Picture Checksum cIdx1 528
HCP Picture Checksum cIdx2 529
HDR 530
HEVC Local APIC Retry Vector 531
HS Invocation Counter 532
IA32 MTRR FIX4K_C0000 High 533
IA32 MTRR FIX4K_C0000 Low 534
IA32 MTRR FIX4K_C8000 High 535
IA32 MTRR FIX4K_C8000 Low 536
IA32 MTRR FIX4K_D0000 High 537
IA32 MTRR FIX4K_D0000 Low 538
IA32 MTRR FIX4K_D8000 High 539
IA32 MTRR FIX4K_D8000 Low 540
IA32 MTRR FIX4K_E0000 High 541
IA32 MTRR FIX4K_E0000 Low 542
IA32 MTRR FIX4K_E8000 High 543
IA32 MTRR FIX4K_E8000 Low 544
IA32 MTRR FIX4K_F0000 High 545
IA32 MTRR FIX4K_F0000 Low 546
IA32 MTRR FIX4K_F8000 High 547
IA32 MTRR FIX4K_F8000 Low 548
IA32 MTRR FIX16K_80000 High 549
IA32 MTRR FIX16K_80000 Low 550
IA32 MTRR FIX16K_A0000 High 551
IA32 MTRR FIX16K_A0000 Low 552
IA32 MTRR FIX64K_00000 High 553
IA32 MTRR FIX64K_00000 Low 554
IA32 MTRR PHYSBASE0 High 555
IA32 MTRR PHYSBASE0 Low 556
IA32 MTRR PHYSBASE1 High 557

IA32 MTRR PHYSBASE1 Low....558
IA32 MTRR PHYSBASE2 High....559
IA32 MTRR PHYSBASE2 Low....560
IA32 MTRR PHYSBASE3 High....561
IA32 MTRR PHYSBASE3 Low....562
IA32 MTRR PHYSBASE4 High....563
IA32 MTRR PHYSBASE4 Low....564
IA32 MTRR PHYSBASE5 High....565
IA32 MTRR PHYSBASE5 Low....566
IA32 MTRR PHYSBASE6 High....567
IA32 MTRR PHYSBASE6 Low....568
IA32 MTRR PHYSBASE7 High....569
IA32 MTRR PHYSBASE7 Low....570
IA32 MTRR PHYSBASE8 High....571
IA32 MTRR PHYSBASE8 Low....572
IA32 MTRR PHYSBASE9 High....573
IA32 MTRR PHYSBASE9 Low....574
IA32 MTRR PHYSMASK0 High....575
IA32 MTRR PHYSMASK0 Low....576
IA32 MTRR PHYSMASK1 High....577
IA32 MTRR PHYSMASK1 Low....578
IA32 MTRR PHYSMASK2 High....579
IA32 MTRR PHYSMASK2 Low....580
IA32 MTRR PHYSMASK3 High....581
IA32 MTRR PHYSMASK3 Low....582
IA32 MTRR PHYSMASK4 High....583
IA32 MTRR PHYSMASK4 Low....584
IA32 MTRR PHYSMASK5 High....585
IA32 MTRR PHYSMASK5 Low....586
IA32 MTRR PHYSMASK6 High....587
IA32 MTRR PHYSMASK6 Low....588
IA32 MTRR PHYSMASK7 High....589
IA32 MTRR PHYSMASK7 Low....590

IA32 MTRR PHYSMASK8 High....591
IA32 MTRR PHYSMASK8 Low....592
IA32 MTRR PHYSMASK9 High....593
IA32 MTRR PHYSMASK9 Low....594
IA Vertices Count....595
IDLE Messaging Register for Blitter Engine....596
IDLE Messaging Register for Media0 Engine....598
IDLE Messaging Register for Media1 Engine....600
IDLE Messaging Register for Render Engine....602
IDLE Messaging Register for VEBox....604
IDLE Messaging Register for Wi-Di....606
Idle Switch Delay....608
Indirect Context Pointer....609
Instruction Parser Mode Register....613
Internal GAM State....617
INTERRUPT LINE....618
Interrupt Mask Register....619
IOBAR....620
L3 Bank Status....621
L3CD Error Status register 1....622
L3 Control Register....624
L3 Control Register1....627
L3 LRA 0....629
L3 LRA 0 GPGPU....630
L3 LRA 1....631
L3 LRA 1 GPGPU....632
L3 LRA 2....633
L3 LRA 2 GPGPU....634
L3 LRA 0 3D....635
L3 LRA 1 3D....636
L3 LRA 2 3D....637
L3 Messaging Register....638
L3 SLM Register....639

L3 SQC register 4 640
L3 SQC registers 1 643
L3 SQC registers 2 647
L3 SQC registers 3 651
LBCF config save msg 655
LBCF DPF Error log register 0 656
LBCF DPF Error log register 1 657
LBCF DPF Error log register 2 658
LBCF DPF Error log register 3 659
LBCF DPF Error log register 4 660
LBCF DPF Error log register 5 661
LBCF DPF Error log register 6 662
LBCF DPF Error log register 7 663
LBCF DPF Error log register 8 664
LBCF DPF Error log register 9 665
LBCF DPF Error log register 10 666
LBCF DPF Error log register 11 667
LBCF DPF Error log register 12 668
LBCF DPF Error log register 13 669
LBCF DPF Error log register 14 670
LBCF DPF Error log register 15 671
LBCF DPF Error log register 16 672
LBCF DPF Error log register 17 673
LBCF DPF Error log register 18 674
LBCF DPF Error log register 19 675
LBCF DPF Error log register 20 676
LBCF DPF Error log register 21 677
LBCF DPF Error log register 22 678
LBCF DPF Error log register 23 679
LBCF DPF Error log register 24 680
LBCF DPF Error log register 25 681
LBCF DPF Error log register 26 682
LBCF DPF Error log register 27 683

LBCF DPF Error log register 28 684
LBCF DPF Error log register 29 685
LBCF DPF Error log register 30 686
LBCF DPF Error log register 31 687
LBCF DPF Error log register 32 688
LBCF DPF Error log register 33 689
LBCF DPF Error log register 34 690
LBCF DPF Error log register 35 691
LBCF DPF Error log register 36 692
LBCF DPF Error log register 37 693
LBCF DPF Error log register 38 694
LBCF DPF Error log register 39 695
LBCF DPF Error log register 40 696
LBCF DPF Error log register 41 697
LBCF DPF Error log register 42 698
LBCF DPF Error log register 43 699
LBCF DPF Error log register 44 700
LBCF DPF Error log register 45 701
LBCF DPF Error log register 46 702
LBCF DPF Error log register 47 703
LBS config bits 704
LEAKAGECOUNTER 705
LEAKAGECOUNTERCTL 706
LEAKAGEWEIGHT1 707
LEAKAGEWEIGHT2 708
LNCF config save msg 709
Load Indirect Base Vertex 710
Load Indirect Instance Count 711
Load Indirect Start Instance 712
Load Indirect Start Vertex 713
Load Indirect Vertex Count 714
LOW 2X FREQUENCY THRESHOLD 715
LPFC control register 716

LTCD Error Injection Register....717
MA....720
Main Graphic Arbiter Error Report....721
Main Graphic Arbiter Error Report 2....725
Main Graphic Arbiter Error Report 3....726
Main Graphic Arbiter Error Report Register....729
MASTER_INT_CTL....730
Master start timer....732
Max Outstanding Pending TLB Requests 0....733
Max Outstanding Pending TLB Requests 1....735
MAX Requests Allowed - GAM....737
MAX Requests Allowed - MFX....739
MAX Requests Allowed - VEBX and BLT....741
MBC Control Register....742
MBDSM....744
MBGSM....745
MD....746
MEDFW_ACK....747
Media 1 TLB Control Register....748
Media 2 TLB Control Register....749
Media forcewake request....750
Media Power Meter Counter....753
Media Power Meter Counter No Clear....754
MEDIARC0COUNTER....755
MEDIARC1COUNTER....756
MEDIARC6COUNTER....757
MEMBOUNDCOUNTER....758
Message Register....759
Messaging Register for GPMunit....762
Messaging Register for MDRBunit....765
Messaging Register for MGSRunit....766
MFC_AVC_CABAC_INSERTION_COUNT....767
MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter....768

MFC Image Status Control 769
MFC Image Status Mask 770
MFC QP Status Count 771
MFD Error Status 772
MFD Picture Parameter 773
MFX_Memory_Latency_Count1 774
MFX0 Context Element Descriptor (High Part) 775
MFX0 Context Element Descriptor (Low Part) 776
MFX0 Context Element Descriptor (Low Part) 777
MFX0 PDP0/PML4/PASID Descriptor (High Part) 778
MFX0 PDP0/PML4/PASID Descriptor (Low Part) 779
MFX0 PDP1 Descriptor Register (High Part) 780
MFX0 PDP1 Descriptor Register (Low Part) 781
MFX0 PDP2 Descriptor Register (High Part) 782
MFX0 PDP2 Descriptor Register (Low Part) 783
MFX0 PDP3 Descriptor Register (High Part) 784
MFX0 PDP3 Descriptor Register (Low Part) 785
MFX1 Context Element Descriptor (High Part) 786
MFX1 Context Element Descriptor (Low Part) 787
MFX1 Context Element Descriptor (Low Part) 788
MFX1 PDP0/PML4/PASID Descriptor (High Part) 789
MFX1 PDP0/PML4/PASID Descriptor (Low Part) 790
MFX1 PDP1 Descriptor Register (High Part) 791
MFX1 PDP1 Descriptor Register (Low Part) 792
MFX1 PDP2 Descriptor Register (High Part) 793
MFX1 PDP2 Descriptor Register (Low Part) 794
MFX1 PDP3 Descriptor Register (High Part) 795
MFX1 PDP3 Descriptor Register (Low Part) 796
MFX Frame BitStream SE/BIN Count 797
MFX Frame Macroblock Count 798
MFX Frame Motion Comp Miss Count 799
MFX Frame Motion Comp Read Count 800
MFX Frame Row-Stored/BitStream Read Count 801

MFX LRA 0 802
MFX LRA 1 803
MFX LRA 2 804
MFX LRA SL1 0 805
MFX LRA SL1 1 806
MFX LRA SL1 2 807
MFX Memory Latency Count2 808
MFX Memory Latency Count3 809
MFX Memory Latency Count4 810
MFX Pipeline Status Flags 811
MFX Slice Performance Count 813
MGGC 814
MGSR2GAM Message Register 817
Misc. Clocking / Reset Control Registers 820
MISC. CTX control register 822
Misc. Reset Control Register 823
Miscellaneous Message Register for Power Managment Unit 824
MISR0 825
MISR1 826
MISR2 827
MISR3 828
MISR4 829
MMIO_INDEX 830
Mode Register for GAB 831
Mode Register for GAC 832
Mode Register for GAFS 833
MSAC 834
MSI_CAPID_MC 837
MTRR Capability Register 0 839
MTRR Capability Register 1 840
MTRR Default Type Register 0 841
MTRR Default Type Register 1 842
MT Virtual Page Address Registers 843

NOP Identification Register 844
P2GCONTROL 845
PAGE_FAULT_MODE 846
Page Directory Pointer Descriptor - PDP0/PML4/PASID 847
Page Directory Pointer Descriptor - PDP1 849
Page Directory Pointer Descriptor - PDP2 850
Page Directory Pointer Descriptor - PDP3 851
Page Request Queue Address Register 0 852
Page Request Queue Address Register 1 853
Page Request Queue Head Register 0 854
Page Request Queue Head Register 1 855
Page Request Queue Tail Register 0 856
Page Request Queue Tail Register 1 857
PAK_Stream-Out Report (Errors) 858
PAK_Stream-Out Report (Warnings) 859
PAK Report Running Status 860
PAT Index 861
PAT Index High 862
PAT Index Low 863
PCBR 864
PCICMD_STS 865
PCU INTERRUPT ENABLE REGISTER 867
PCU INTERRUPT IDENTITY REGISTER 868
PCU INTERRUPT MASK REGISTER 869
PCU INTERRUPT STATUS REGISTER 870
Pending Head Pointer Register 871
Plink G2H Spare 873
PM_PWR_CLK_STATE 874
PMCAPID 878
PMCS 880
POWER_WELL_SS0_SIG1 881
POWER_WELL_SS0_SIG2 886
POWER_WELL_SS1_SIG1 888

POWER_WELL_SS1_SIG2 893
Power Context Save 895
Power context Save Register for LPFC 897
Power Context Save request 898
POWERDOWN_STATE 899
POWERDOWN_WAIT1 901
POWERDOWN_WAIT2 902
Power Down/Up Control 903
Power Enable stagger control 904
Power Meter Weight for gti_idle_cz 905
Power Meter Weight for gti_idle_gs 906
Power Meter Weight for media/render 907
POWERUP_STATE 908
POWERUP_WAIT1 910
POWERUP_WAIT2 911
PPGTT Page Fault Data Registers 912
Predicate Rendering Data Result 913
Predicate Rendering Data Result 1 914
Predicate Rendering Data Result 2 915
Predicate Rendering Data Storage 916
Predicate Rendering Temporary Register0 917
Predicate Rendering Temporary Register1 918
Previous Idle/Busy/Avg Count for Freq Down Recommendation 919
Previous Idle/Busy/Avg Count for Freq Up Recommendation 920
Primitives Generated By VF 921
Private PAT 922
Private PAT 923
PS Depth Count 924
PS Depth Count for Slice0 925
PS Depth Count for Slice1 926
PS Depth Count for Slice2 927
PS Depth Count for Slice3 928
PS Invocation Count 929

PS Invocation Count for Slice0 930
PS Invocation Count for Slice1 931
PS Invocation Count for Slice2 932
PS Invocation Count for Slice3 933
PTBR_PAGE_POOL_OOM_EVENT_REGISTER 934
PTBR Page Pool Size on Out Of Memory 935
PTE SW Fault Repair High 936
PTE SW Fault Repair Low 937
Punit to Gunit Message 938
PWRCTXSAVE Message Register for Power Managment Unit 940
PWRMTR_WT1_EEVT1TO4 941
PWRMTR_WT1_EEVT5TO8 942
PWRMTR_WT1_EEVT9TO12 943
PWRMTR_WT1_EEVT13TO16 944
PWRMTR_WT1_EEVT17TO20 945
PWRMTR_WT1_EEVT21TO24 946
PWRMTR_WT1_EEVT25TO28 947
PWRMTR_WT1_EEVT29TO32 948
PWRMTR_WT1_EEVT33TO36 949
PWRMTR_WT1_EEVT37TO40 950
PWRMTR_WT1_EEVT41TO44 951
PWRMTR_WT1_EEVT45TO48 952
PWRMTR_WT1_EEVT49TO52 953
PWRMTR_WT1_EEVT53TO56 954
PWRMTR_WT1_EEVT57TO58 955
PWRMTR_WT1_MEVT1TO4 956
PWRMTR_WT1_MEVT5TO8 957
PWRMTR_WT1_MEVT9TO12 958
PWRMTR_WT1_MEVT13TO16 959
PWRMTR_WT1_MEVT17TO20 960
PWRMTR_WT1_MEVT21TO24 961
PWRMTR_WT1_MEVT25TO28 962
PWRMTR_WT1_MEVT29TO32 963

PWRMTR_WT1_MEVT33TO36 964
PWRMTR_WT1_REVT1TO4 965
PWRMTR_WT1_REVT5TO8 966
PWRMTR_WT1_REVT9TO12 967
PWRMTR_WT1_REVT13TO16 968
PWRMTR_WT1_REVT17TO20 969
PWRMTR_WT1_REVT21TO24 970
PWRMTR_WT1_REVT25TO28 971
PWRMTR_WT1_REVT29TO32 972
PWRMTR_WT1_REVT33TO36 973
PWRMTR_WT1_REVT37TO40 974
PWRMTR_WT1_REVT41TO44 975
PWRMTR_WT1_REVT45TO48 976
PWRMTR_WT1_REVT49TO52 977
PWRMTR_WT1_REVT53TO56 978
PWRMTR_WT1_REVT57TO60 979
PWRMTR_WT1_REVT61TO64 980
PWRMTR_WT1_REVT65TO68 981
PWRMTR_WT1_REVT69TO70 982
PWRMTR_WT2_EEVT1TO4 983
PWRMTR_WT2_EEVT5TO8 984
PWRMTR_WT2_EEVT9TO12 985
PWRMTR_WT2_EEVT13TO16 986
PWRMTR_WT2_EEVT17TO20 987
PWRMTR_WT2_EEVT21TO24 988
PWRMTR_WT2_EEVT25TO28 989
PWRMTR_WT2_EEVT29TO32 990
PWRMTR_WT2_EEVT33TO36 991
PWRMTR_WT2_EEVT37TO40 992
PWRMTR_WT2_EEVT41TO44 993
PWRMTR_WT2_EEVT45TO48 994
PWRMTR_WT2_EEVT49TO52 995
PWRMTR_WT2_EEVT53TO56 996

PWRMTR_WT2_EEVT57TO58 997
PWRMTR_WT2_REVT5TO8 998
PWRMTR_WT2_REVT33TO36 999
PWRMTR_WT2_REVT37TO40 1000
PWRMTR_WT2_REVT41TO44 1001
PWRMTR_WT2_REVT49TO52 1002
PWRMTR_WT2_REVT53TO56 1003
PWRMTR_WT2_REVT65TO68 1004
PWRMTR_WT2_REVT69TO70 1005
PWRMTR_WT3_EEVT1TO4 1006
PWRMTR_WT3_EEVT5TO8 1007
PWRMTR_WT3_EEVT9TO12 1008
PWRMTR_WT3_EEVT13TO16 1009
PWRMTR_WT3_EEVT17TO20 1010
PWRMTR_WT3_EEVT21TO24 1011
PWRMTR_WT3_EEVT25TO28 1012
PWRMTR_WT3_EEVT29TO32 1013
PWRMTR_WT3_EEVT33TO36 1014
PWRMTR_WT3_EEVT37TO40 1015
PWRMTR_WT3_EEVT41TO44 1016
PWRMTR_WT3_EEVT45TO48 1017
PWRMTR_WT3_EEVT49TO52 1018
PWRMTR_WT3_EEVT53TO56 1019
PWRMTR_WT3_EEVT57TO58 1020
PWRMTR_WT3_REVT5TO8 1021
PWRMTR_WT3_REVT33TO36 1022
PWRMTR_WT3_REVT37TO40 1023
PWRMTR_WT3_REVT41TO44 1024
PWRMTR_WT3_REVT49TO52 1025
PWRMTR_WT3_REVT53TO56 1026
PWRMTR_WT3_REVT65TO68 1027
PWRMTR_WT3_REVT69TO70 1028
PWRMTREVTORE0 1029

PWRMTREVTORE1 1030
PWRMTREVTORM0 1031
PWRMTREVTORM1 1032
PWRMTREVTORR0 1033
PWRMTREVTORR1 1034
PWRMTREVTORR2 1035
RAM Clock Gating Control 1 1036
RAM Clock Gating Control 1 1042
RAM Clock Gating Control 2 1048
RAM Clock Gating Control 2 1053
RAM Clock Gating Control 3 1058
RC6 Wake Rate Limit 1061
RCC LRA 0 1062
RCC LRA 1 1063
RCC Virtual page Address Registers 1064
RC Evaluation Interval 1065
RC Idle Hysteresis 1066
RCS_PREEMPTION_HINT 1067
RCS_PREEMPTION_HINT_UDW 1069
RCS Batch Buffer State Register 1070
RCS Context Preemption Hint 1071
RC Wake Counter 1072
RCZ Virtual Page Address Registers 1073
Ready Bit Vector 0 for TLBPEND registers 1074
Ready Bit Vector 1 for TLBPEND registers 1075
Render C State Control 1 1076
Render forcewake acknowledge 1077
Render forcewake request 1078
Render Geyserville Mode Control 1 1081
Render Mode Register for Software Interface 1084
Render Performance Status 1 1088
Render Performance Status Register 1089
Render Power Clock State Register 1090

Render Power Meter Counter 1092
Render Power Meter Counter No Clear 1093
Render Promotion Timer - RC6 1094
RENDERRC0COUNTER 1095
RENDERRC1COUNTER 1096
RENDERRC6COUNTER 1097
Render TLB Control Register 1098
Render Watchdog Counter 1099
Render Watchdog Counter Threshold 1100
Reported BitRateControl parameter Mask 1101
Reported BitRateControl parameter Status 1103
Reported Bitstream Output Bit Count for Syntax Elements Only Register 1105
Reported Bitstream Output Byte Count per Frame Register 1106
Reported Bitstream Output CABAC Bin Count Register 1107
Reported Final Bitstream Byte Count 1108
Reported Frame Zero Padding Byte Count 1109
Reported Timestamp Count 1110
Reset Flow Control Messages 1112
RESET Messaging Register for Clocking Unit 1115
Resource Streamer Context Offset 1117
Resource Streamer Preemption Status 1118
Restored Timestamp LSDW 1120
Restored timestamp MSDW 1121
RID_CC 1122
RING_BUFFER_HEAD_PREEMPT_REG 1123
Ring Buffer Control 1125
Ring Buffer Current Context ID Register 1128
Ring Buffer Head 1129
Ring Buffer Start 1131
Ring Buffer Tail 1132
Root Table Address Pointer Value First 31_0 1134
Root Table Address Pointer Value Second 31_0 1135
RP Decrease Limit 1136

RP Down Timeout 1137
RP Downwards Evaluation Interval 1138
RP Increase Limit 1139
RP Normal Software Frequency Request 1140
RP Software Frequency Request Hysteresis 1141
RP Upwards Evaluation Interval 1142
RP Video Turbo Software Frequency Request 1143
RS_PREEMPT_STATUS_UDW 1144
RS Preemption Hint 1145
RS Preemption Hint UDW 1146
Sampler control register 1147
SAMPLER Mode Register 1148
SAMPLER READ DATA 1149
Save Timer 1150
SB_ADDRESS 1151
SB_DATA 1152
SB_REQ_TRIGGER 1153
SCPD0 1155
SCRATCH1 1156
SCRATCH for LNCFunit 1157
Scratch Register 1 1158
Scratch Register 2 1159
Second Buffer Size 1160
Second Level Batch Buffer Head Pointer Preemption Register 1161
Second Level Batch Buffer Head Pointer Register 1163
Second Level Batch Buffer State Register 1165
Second Level Batch Buffer Upper Head Pointer Preemption Register 1167
Second Level Batch Buffer Upper Head Pointer Register 1168
Semaphore Polling Interval on Wait 1169
SSID_SID 1170
Staggered EU/SAMPLER PAUSE on Frequency Change 1171
Storage 1 1172
Storage 2 1173

Storage 3 1174
Storage 4 1175
Storage 5 1176
Storage 6 1177
Storage 7 1178
Storage 8 1179
Storage 9 1180
Stream Output Num Primitives Written Counter 1181
Stream Output Primitive Storage Needed Counters 1182
Stream Output Write Offsets 1183
SWF1 1184
SWF2 1185
SWF3 1186
SWF4 1187
SWF5 1188
SWF6 1189
SWF7 1190
SWF8 1191
SWF9 1192
SWF10 1193
SWF11 1194
SWF12 1195
SWF13 1196
SWF14 1197
SWF15 1198
SWF16 1199
SWF17 1200
SWF18 1201
SWF19 1202
SWF20 1203
SWF21 1204
SWF22 1205
SWF23 1206

SWF24 1207
SWF25 1208
SWF26 1209
SWF27 1210
SWF28 1211
SWF29 1212
SWF30 1213
SWF31 1214
SWF32 1215
SWF33 1216
SWF34 1217
SWF35 1218
SWF36 1219
SWSMISCI 1220
Thread Dispatched Count Register 1221
Thread Faulted Count Register 1222
Thread Fault Status Register 0 1223
Thread Fault Status Register 1 1224
Thread Load Status Register 0 1225
Thread Load Status Register 1 1226
Thread Mode Register 1227
Thread Restart Control Register 1230
TILECTL 1231
TiledResources Invalid Tile Detection Register 1232
Tiled Resources Translation Table Control Registers 1233
TiledResources VA Detection Registers 1234
Tiled Resources VA Translation Table L3 ptr - DW0 1235
Tiled Resources VA Translation Table L3 ptr - DW1 1236
TLB_RD_ADDRESS Register 1237
TLB_RD_DATA0 Register 1238
TLB_RD_DATA1 Register 1239
Transcode Attack Status Register 1240
TRNULLDETCT 1241

Turbo Media Control 1242
Unblock Message Act to Busy Detection Timer 1243
Unit Level Clock Gating Control 1 1244
Unit Level Clock Gating Control 1 1250
Unit Level Clock Gating Control 2 1256
Unit Level Clock Gating Control 2 1262
Unit Level Clock Gating Control 3 1268
Unit Level Clock Gating Control 3 1274
Unit Level Clock Gating Control 4 1280
Unit Level Clock Gating Control 4 1286
Unit Level Clock Gating Control 5 1292
Unit Level Clock Gating Control 5 1298
Unit Level Clock Gating Control 6 1304
Unit Level Clock Gating Control 6 1309
Unit Level Clock Gating Control 7 1315
Unit Level Clock Gating Control 7 1319
Unit Level Clock Gating Control 8 1325
Unit Level Clock Gating Control 9 1331
URB Context Offset 1332
Valid Bit Vector 0 for CVS 1333
Valid Bit Vector 0 for L3 1334
Valid Bit Vector 0 for MFX 1335
Valid Bit Vector 0 for MFX SL1 1336
Valid Bit Vector 0 for MTTLB 1337
Valid Bit Vector 0 for MTVICTLB 1338
Valid Bit Vector 0 for RCC 1339
Valid Bit Vector 0 for RCCTLB 1340
Valid Bit Vector 0 for RCZTLB 1341
Valid Bit Vector 0 for TLBPEND registers 1342
Valid Bit Vector 0 for VEBX 1343
Valid Bit Vector 0 for WIDI 1344
Valid Bit Vector 0 for Z 1345
Valid Bit Vector 1 for CVS 1346

Valid Bit Vector 1 for L3....1347
Valid Bit Vector 1 for MFX....1348
Valid Bit Vector 1 for MFX SL1....1349
Valid Bit Vector 1 for MTTLB....1350
Valid Bit Vector 1 for MTVICTLB....1351
Valid Bit Vector 1 for RCC....1352
Valid Bit Vector 1 for RCCTLB....1353
Valid Bit Vector 1 for RCZTLB....1354
Valid Bit Vector 1 for TLBPEND registers....1355
Valid Bit Vector 1 for VEBX....1356
Valid Bit Vector 1 for WIDI....1357
Valid Bit Vector 1 for Z....1358
Valid Bit Vector 2 for CVS....1359
Valid Bit Vector 2 for GAB....1360
Valid Bit Vector 2 for L3....1361
Valid Bit Vector 2 for MFX....1362
Valid Bit Vector 2 for MFX SL1....1363
Valid Bit Vector 2 for RCC....1364
Valid Bit Vector 2 for Z....1365
Valid Bit Vector 3 for CVS....1366
Valid Bit Vector 3 for L3....1367
Valid Bit Vector 3 for MFX....1368
Valid Bit Vector 3 for MFX SL1....1369
Valid Bit Vector 3 for RCC....1370
Valid Bit Vector 3 for Z....1371
Valid Bit Vector 4 for L3....1372
Valid Bit Vector 4 for MFX....1373
Valid Bit Vector 4 for MFX SL1....1374
Valid Bit Vector 4 for RCC....1375
Valid Bit Vector 4 for Z....1376
Valid Bit Vector 5 for L3....1377
Valid Bit Vector 5 for MFX....1378
Valid Bit Vector 5 for MFX SL1....1379

Valid Bit Vector 5 for RCC 1380
Valid Bit Vector 5 for Z 1381
Valid Bit Vector 6 for L3....1382
Valid Bit Vector 6 for MFX....1383
Valid Bit Vector 6 for MFX SL1 1384
Valid Bit Vector 6 for RCC 1385
Valid Bit Vector 6 for Z 1386
Valid Bit Vector 7 for L3....1387
Valid Bit Vector 7 for MFX....1388
Valid Bit Vector 7 for MFX SL1 1389
Valid Bit Vector 7 for RCC 1390
Valid Bit Vector 7 for Z 1391
Valid Bit Vector 8 for L3....1392
Valid Bit Vector 8 for Z 1393
Valid Bit Vector 9 for L3....1394
Valid Bit Vector 9 for Z 1395
Valid Bit Vector 10 for L3 1396
Valid Bit Vector 10 for Z 1397
Valid Bit Vector 11 for L3 1398
Valid Bit Vector 11 for Z 1399
Valid Bit Vector 12 for L3 1400
Valid Bit Vector 12 for Z 1401
Valid Bit Vector 13 for L3 1402
Valid Bit Vector 13 for Z 1403
Valid Bit Vector 14 for L3 1404
Valid Bit Vector 14 for Z 1405
Valid Bit Vector 15 for L3 1406
Valid Bit Vector 15 for Z 1407
Valid Bit Vector 16 for L3 1408
Valid Bit Vector 17 for L3 1409
Valid Bit Vector 18 for L3 1410
Valid Bit Vector 19 for L3 1411
Valid Bit Vector 20 for L3 1412

Valid Bit Vector 21 for L3 1413
Valid Bit Vector 22 for L3 1414
Valid Bit Vector 23 for L3 1415
Valid Bit Vector for VLF 1416
Valid Bit Vector for VLFSL1 1417
VC 1418
VCES Idle Switch Delay 1419
VCID 1420
VCS_PREEMPTION_HINT 1421
VCS_PREEMPTION_HINT_UDW 1423
VCS Context ID Preemption Hint 1424
VCS Context Sizes 1425
VCS Context Timestamp Count 1426
VCS Counter for the bit stream decode engine 1427
VCS Error Identity Register 1428
VCS Error Mask Register 1429
VCS Error Status Register 1430
VCS Execute Condition Code Register 1431
VCS General Purpose Register 1432
VCS Hardware Status Mask Register 1433
VCS IDLE Max Count 1434
VCS Idle Switch Delay 1435
VCS Instruction Parser Mode Register 1436
VCS Interrupt Mask Register 1438
VCS Mode Register for Software Interface 1439
VCS Reported Timestamp Count 1441
VCS Reset Control Register 1442
VCS Ring Buffer Next Context ID Register 1443
VCS Semaphore Polling Interval on Wait 1444
VCS Threshold for the counter of bit stream decode engine 1445
VCW Clock Count 1446
VCW Internal Latency 1447
VCW Min Max Latency 1448

VCW Total Latency....1449
VCW XY position....1450
VEBOX TLB Control Register....1451
VEBX Context Element Descriptor (High Part)....1452
VEBX Context Element Descriptor (Low Part)....1453
VEBX Context Element Descriptor (Low Part)....1454
VEBX LRA 0....1455
VEBX LRA 1....1456
VEBX PDP0/PML4/PASID Descriptor (High Part)....1458
VEBX PDP0/PML4/PASID Descriptor (Low Part)....1459
VEBX PDP1 Descriptor Register (High Part)....1460
VEBX PDP1 Descriptor Register (Low Part)....1461
VEBX PDP2 Descriptor Register (High Part)....1462
VEBX PDP2 Descriptor Register (Low Part)....1463
VEBX PDP3 Descriptor Register (High Part)....1464
VEBX PDP3 Descriptor Register (Low Part)....1465
VECS_PREEMPTION_HINT....1466
VECS Context ID Preemption Hint....1468
VECS Context Timestamp Count....1469
VECS Counter for the Video Enhancement Engine....1470
VECS Error Identity Register....1471
VECS Error Mask Register....1472
VECS Error Status Register....1473
VECS General Purpose Register....1474
VECS Hardware Status Mask Register....1475
VECS IDLE Max Count....1476
VECS Instruction Parser Mode Register....1477
VECS Interrupt Mask Register....1478
VECS Mode Register for Software Interface....1479
VECS PREEMPTION HINT UDW....1481
VECS Reported Timestamp Count....1482
VECS Reset Control Register....1483
VECS Semaphore Polling Interval on Wait....1484

VECS Threshold for the Counter of Video Enhancement Engine 1485
VEO Current Pipe 0 XY Register....1486
VEO DN Pipe 0 XY Register 1487
VEO DN Pipe 1 XY Register 1488
VEO DV Count Register....1489
VEO DV Hold Register....1490
VEO Previous Pipe 0 XY Register....1493
VF Scratch Pad....1494
VFW Credit Count Register 1496
VIC Virtual page Address Registers 1497
VIDEOBUSYCOUNTER 1498
Video Enhancement Mode Register....1499
Video Mode Register....1502
VS Invocation Counter 1505
Wait For Event and Display Flip Flags Register....1506
Wait For Event and Display Flip Flags Register 1....1511
Walkers Fault Register 1517
WD_WNIC_MSG_ADDR 1518
WGBOX State Arbitration Priority Control 1519
WIDI LRA 0....1520
WIDI LRA 1....1521
WIDI TLB Control Register 1522
WRID_VALID_REG0 1523
WRID_VALID_REG1 1524
WRID_VALID_REG2 1525
Write Watermark 1526
ZTLB LRA 0....1527
ZTLB LRA 1....1528

Advanced Features Length and Capabilities

AFLC - Advanced Features Length and Capabilities

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x03060013
Size (in bits):	32
Address:	000A4h

FLR capability advertisement

DWord	Bit	Description
0	31:26	**RESERVED** Default Value: 00h Access: RO Reserved
	25	**FLR_CAP** Default Value: 1b Access: RO Function Level Reset Capability (FLR_CAP): 0: Function Level Reset is not supported 1: Function Level Reset is supported
	24	**TP_CAP** Default Value: 1b Access: RO Transactions Pending Capability (TP_CAP): 0: Transactions Pending bit is not supported 1: Transactions Pending bit is supported
	23:16	**LENGTH** Default Value: 06h Access: RO Advanced Features Structure Length(LENGTH): The Advanced Features capability structure is 6bytes long.
	15:8	**NXT_PTR** Default Value: 00h Access: R/W Once Next Pointer (NXT_PTR):

AFLC - Advanced Features Length and Capabilities

		Points to the next item in the list (B0=Vendor Capabilities ID). This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset. Write once so capabilities list can be changed if needed.
	7:0	**CAP_ID** Default Value: 13h Access: RO Capability Identifier (CAP_ID): A value of 13h identifies that this PCI Function is capable of Advanced Features.

Advanced Scheduler Reset Request Messages

ASSRREQ - Advanced Scheduler Reset Request Messages

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0810Ch

Hardware (CS, VCS) initiated Advanced Scheduler reset request messages.

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:4	**Reserved** Access: RO Reserved
	3	**VINunit cmfxrst reset request message (2nd Vbox)** Access: R/W Set CMFX Reset Request Message from the VINunit in 2nd Vbox: '1' : CMFX Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : CMFX Reset Not Requested
	2	**VINunit cmfxrst Reset Request message** Access: R/W Set CMFX Reset Request Message from the VINunit: '1' : CMFX Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : CMFX Reset Not Requested
	1	**Render AS Reset Request Message** Access: R/W Set Render AS Reset Request Message from the CSunit: '1' : Render AS Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : Render AS Reset Not Requested

ASSRREQ - Advanced Scheduler Reset Request Messages

	0	**Media AS Reset Request Message** Access: R/W Set Media AS Reset Request Message from the VCSunit: '1' : Media AS Reset Requested - This bit is cleared by the CP upon completion of the reset request '0' : Media AS Reset Not Requested

Aggregate_Perf_Counter_A31

OAPERF_A31 - Aggregate_Perf_Counter_A31	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028F8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A31

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A32

OAPERF_A32 - Aggregate_Perf_Counter_A32

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02900h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A32

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A33

OAPERF_A33 - Aggregate_Perf_Counter_A33	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02904h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A33

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A34

OAPERF_A34 - Aggregate_Perf_Counter_A34	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02908h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A34

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate_Perf_Counter_A35

OAPERF_A35 - Aggregate_Perf_Counter_A35	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0290Ch
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A35

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A1

OAPERF_A1 - Aggregate Perf Counter A1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02808h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A1. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	Considerations: This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A5

OAPERF_A5 - Aggregate Perf Counter A5	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02828h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A5. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A7

OAPERF_A7 - Aggregate Perf Counter A7	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02838h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A7. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A8

OAPERF_A8 - Aggregate Perf Counter A8	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02840h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A8. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A9

OAPERF_A9 - Aggregate Perf Counter A9	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02848h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A9. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A10

OAPERF_A10 - Aggregate Perf Counter A10	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02850h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A10. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A11

OAPERF_A11 - Aggregate Perf Counter A11	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02858h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A11. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A12

OAPERF_A12 - Aggregate Perf Counter A12	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02860h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A12. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A13

OAPERF_A13 - Aggregate Perf Counter A13	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02868h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A13. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A14

OAPERF_A14 - Aggregate Perf Counter A14	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02870h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A14. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A15

OAPERF_A15 - Aggregate Perf Counter A15	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02878h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A15. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A16

OAPERF_A16 - Aggregate Perf Counter A16	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02880h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A16. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A17

OAPERF_A17 - Aggregate Perf Counter A17	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02888h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A17. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A18

OAPERF_A18 - Aggregate Perf Counter A18	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02890h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A9. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A21

OAPERF_A21 - Aggregate Perf Counter A21	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028A8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A21. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A22

OAPERF_A22 - Aggregate Perf Counter A22	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028B0h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A22. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A23

OAPERF_A23 - Aggregate Perf Counter A23	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028B8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A23. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A24

OAPERF_A24 - Aggregate Perf Counter A24	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028C0h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A24. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A25

OAPERF_A25 - Aggregate Perf Counter A25	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028C8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A25. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A26

OAPERF_A26 - Aggregate Perf Counter A26	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028D0h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A26. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A27

OAPERF_A27 - Aggregate Perf Counter A27	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028D8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A27. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A28

OAPERF_A28 - Aggregate Perf Counter A28	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028E0h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A28. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A29

OAPERF_A29 - Aggregate Perf Counter A29	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028E8h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A29. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Aggregate Perf Counter A30

OAPERF_A30 - Aggregate Perf Counter A30	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	028F0h
Valid Projects:	[CHV, BSW]

This register reflects the count value of the OA Performance counter A30. DefaultValue="00000000h"

DWord	Bit	Description
0	31:0	**Considerations** This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

AFCTLSTS

AFCTLSTS - AFCTLSTS

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	000A8h

FLR control
Advanced Feature Status

DWord	Bit	Description
0	15:9	**Reserved** Default Value: 00h Access: RO Reserved (RSVD)
	8	**TP** Default Value: 0b Access: RO Transaction Pending (TP): 1: The Function has issued one or more non-posted transactions which have not been completed, including non-posted transactions that a target has terminated with Retry. 0: All non-posted transactions have been completed.
	7:1	**Reserved** Default Value: 00h Access: RO Reserved (RSVD)
	0	**INIT_FLR** Default Value: 0b Access: R/W Set Initiate Function Level Reset (INIT_FLR): A write of 1b initiates Function Level Reset (FLR). FLR requirements are defined in the PCI Express Base Specification. Registers and state information that do not apply to conventional PCI are exempt from the FLR requirements given there. Once written 1, FLR will be initiated. During FLR, a read will return 1's since device 2 reads abort. Once FLR completes, hardware will clear the bit to 0.

All Engine Fault Register

FAULT_REG - All Engine Fault Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04094h	

DWord	Bit	Description
0	31:1	**All Engine Fault Register** Default Value: 00000000000000000000000000000000b Access: R/W Bit[31:15]: Reserved. Bit[14:12]: Engine ID: 000b - GFX. 001b - MFX0. 010b - MFX1. 011b - VEBX. 100b - BLT. 110b - WIDI. Bit[11]: Reserved. Bit[10:3]: SRCID of Fault. This is the Source ID of the unit that requested the cycle that generated the First Page fault for this engine. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW. Bit[2:1]: Fault Type (GFX_FT): Type of Fault recorded: 00b - Invalid PTE Fault. 01b - Invalid PDE Fault. 10b - Invalid PDPE Fault. 11b - Invalid PML4E Fault. This value is locked and not updated on subsequent faults, until the valid bit of this register is cleared by SW. All bits are only valid with bit[0]=1.
	0	**Valid Bit** Default Value: 0b Access: R/W This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which also clears the other fields.

ARAT CUTRIG HI

ARAT_CUTRIG_HI - ARAT CUTRIG HI		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A1B4h	
DWord	**Bit**	**Description**
0	31:0	**ARAT TRIG HI** Access: RO [31:0]: 63:32 of (ARAT_TDELTA + TSC)[63:32] when A174[0] transitions to a one, or when A174[1:0]=11 and TSC crosses over ARAT_TRIG

ARAT CUTRIG LO

ARAT_CUTRIG_LO - ARAT CUTRIG LO		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A1B0h	
DWord	**Bit**	**Description**
0	31:3	**ARAT TRIG LO** Access: RO 31:3 of (ARAT_TDELTA + TSC)[31:3] when A174[0] transitions to a one, or when A174[1:0]=11 and TSC crosses over ARAT_TRIG
	2	**Reserved** Access: RO
	1:0	**ARAT TRIG Mode** Access: RO 0xA174[1:0] when A174[0] transitions to a one, or when A174[1:0]=11 and TSC crosses over ARAT_TRIG

ARAT Delta (LSB)

ARAT_TDELTA_LOW - ARAT Delta (LSB)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A174h	

DWord	Bit	Description
0	31:3	**Lower Bits of Delta Time for ARAT** Access: R/W Low Bits [31:3] of Delta Time, in 80ns increments(LSB would be in 10ns increments, if it went down to zero)
	2	**Reserved** Access: RO
	1	**ARAT Mode** Access: R/W 0b : One-Shot Mode (default) 1b : Periodic Mode
	0	**ARAT Enable** Access: R/W 0b : ARAT Disabled (default) 1b : ARAT Enabled

ARAT Delta (MSB)

ARAT_TDELTA_HIGH - ARAT Delta (MSB)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A170h	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO
	23:0	**Upper Bits of Delta Time for ARAT** Access: R/W High Bits [55:32] of Delta Time, in 80ns increments

ARAT POST CU BUSY

ARAT_POSTCUBUSY - ARAT POST CU BUSY		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A17Ch	
DWord	**Bit**	**Description**
0	31:3	**ARAT PostThreshold** Access: R/W PostThreshold, in 80ns increments to prevent short gfx_clockstartreq. If current_TSC - ARAT_CUTRIG > PostThreshold, show GFX as busy.
	2:0	**Reserved** Access: RO

ARAT PRE CU BUSY

ARAT_PRECUBUSY - ARAT PRE CU BUSY		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A178h	
DWord	**Bit**	**Description**
0	31:3	**ARAT PreThreshold** Access: R/W PreThreshold, in 80ns increments to prevent short gfx_clockstartreq. If ARAT_CUTRIG minus current_TSC < PreThreshold, show GFX as busy.
	2:0	**Reserved** Access: RO

Arbiter Control Register

GARBCNTLREG - Arbiter Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x29124100 CHV, BSW
Size (in bits):	32
Address:	0B004h

DWord	Bit	Description
0	31	**Reserved**
	30	**Disables hashing function** Access: R/W Disables hashing function to generate bank_id[1:0] for L3$ bank accessing, and forces the use of address[7:6] for bank_id[1:0].0: (default) Hash function enabled to generate L3$ bank IDs. 1: L3$ address[7:6] used as L3$ bank IDs. Incf_csr_l3bankidhashdis. (This bit needs to set corresponding bit lpfcon_csr_l3bankidhashdis in LPFC.)
	29:28	**Arbitration priority order between RCC and MSC** Default Value: 10b Access: R/W Arbitration priority order between RCC and MSC. 00b/11b: Invalid; default setting used. 10b: Default setting; RCC MSC (i.e., MSC has higher priority). 01b: RCC MSC (i.e., RCC has higher priority). Incf_csr_rcc_msc_pri[1:0].
	27:22	**Arbitration priority order between RCZ, STC, and HIZ** Default Value: 100100b Access: R/W Arbitration priority order between RCZ, STC, and HIZ. 100100b: Default setting; RCZ STC HIZ. (i.e., RCZ has lowest priority; HIZ has highest priority). 100001b: RCZ ; HIZ ; STC. 011000b: STC ; RCZ ; HIZ. 010010b: STC ; HIZ ; RCZ. 001001b: HIZ ; RCZ ; STC. 000110b: HIZ ; STC ; RCZ. Note: Others settings are invalid, and result in use of default. Incf_csr_rcz_stc_hiz_pri[5:0].

GARBCNTLREG - Arbiter Control Register

	21:19	**Write data port arbitration priority between Z client writes and L3$ evictions** Default Value: 010b Access: R/W Z Max Write Request Limit Count (GFXC_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (both Slice 0 and 1). Minimum count value must be = 1. Incf_csr_wdpagapz[2:0].
	18:16	**Write data port arbitration priority between C client writes and Z/L3$ writes/evictions** Default Value: 010b Access: R/W C Max Request Limit Count (GFXZ_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (both Slice 0 and 1). Minimum count value must be = 1. Incf_csr_wdpagapc[2:0].
	15	**Reserved** Access: RO
	14:12	**L3 Max Write Request Limit Count** Default Value: 100b Access: R/W L3 Max Write Request Limit Count (GFXL3_MRLC). This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1. Incf_csr_wdpagapl3[2:0].
	11:9	**Reserved** Access: RO
	8	**GAPs_fixarb_en** Default Value: 1b Access: R/W Incf_csr_gaps_fixarb_en.
	6:0	**Reserved** Project: CHV, BSW Access: RO

Arbiter Mode Control Register

ARB_MODE - Arbiter Mode Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04030h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000000000000000b Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**Extra Register Bit 15** Default Value: 0b Access: R/W Bit 15 toggles (XOR) the meaning of Per Client Write Drop Enables (Register 40b4); If 0, drop per client happens as stated in register 40b4 definition; If 1, the meaning changes, and a 1 on a bit in register 40b4 means dont drop while 0 means drop. In this case, the default (for clients not included in 40b4) will be drop enabled.
	14	**Extra Register Bit 14** Default Value: 0b Project: CHV, BSW Access: R/W PD load disable - When this bit is set, the PD load is disabled for GFX/MFX0/MFX1. A-step: Default Value: 0 B-step: Default Value: 0 - Bug ID: 1905990 Future steppings can have value 1.
	13	**DC GDR** Default Value: 0b Access: R/W
	12	**HIZ GDR** Default Value: 0b Access: R/W
	11	**STC GDR** Default Value: 0b Access: R/W

ARB_MODE - Arbiter Mode Control Register

10	**BLB GDR** Default Value: 0b Access: R/W
9	**GAM PD GDR** Default Value: 0b Access: R/W
8	**Extra Register Bit 8** Default Value: 0b Access: R/W **Description** Snoop Override [CHV, BSW Only] 0 - No override (default) 1 - Snoop is set for all accesses to memory
7:6	**Cacheability Attribute Override** Default Value: 00b Access: R/W 00b No override. 01b UC (LLC/eLLC) - Allocation age is don't care. 10b WT in LLC/eLLC - Aged is 3. 11b WB in LLC/eLLC - Aged is 3. The above conditions apply for the following conditions only: 1. Register overwrite except for GTT, CFG and L3 coherent wcil cycles 2. Read- GTTRD, CFGRD 3. Write- GTTWR, CFGWR, DMWR (with gam_ci_wcoherenttype[2:0]="001" WCIL* w/self snoop)
5	**Extra Register Bit 5** Default Value: 0b Access: R/W Reserved.
4	**VMC GDR Enable** Default Value: 0b Access: R/W When this bit is set, data requested from the VMC client is generated by the GDR Algorithm.
3	**Texture Cache (MT) GDR Enable Bit** Default Value: 0b Access: R/W When this bit is set, data requested from the Texture Cache (MT) client is generated by the GDR algorithm.

ARB_MODE - Arbiter Mode Control Register

	2	**Depth (RCZ) Cache GDR Enable bit** Default Value: 0b Access: R/W Depth Cache GDR enable bit. Project: All. Format: U1. When this bit is set, data requested from the Depth Cache client is generated by the GDR algorithm (See GDR algorithm in xxx section).
	1	**Color Cache (RCC) GDR Enable Bit** Default Value: 0b Access: R/W When this bit is set, data requested from the Color Cache (RCC) client is generated by the GDR algorithm.
	0	**GTT Accesses GDR** Default Value: 0b Access: R/W When this bit is enabled along with the Client's GDR bit, PPGTT and GGTT requests for this memory access are also tagged as GDR to SQ.

ASLS

ASLS - ASLS

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	000FCh

ASL Storage.

This software scratch register only needs to be read/write accessible. The exact bit register usage must be worked out in common between System BIOS and driver software, but storage for switching/indicating up to 6 devices is possible with this amount.
For each device, the ASL control method with require two bits for _DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for _DGS (enable/disable requested), and two bits for _DCS (enabled now/disabled now, connected or not).

DWord	Bit	Description
0	31:0	**SCRATCH** Default Value: 00000000h Access: R/W This register provides a means for the BIOS to communicate with the driver. This definition of this scratch register is worked out in common between System BIOS and driver software. Storage for up to 6 devices is possible. For each device, the ASL control method requires two bits for _DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for _DGS (enable/disable requested), and two bits for DCS (enabled now/disabled now, connected or not).

ASYNC_SLICE_COUNT

ASYNC_SLICE_COUNT - ASYNC_SLICE_COUNT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000280
Size (in bits):	32
Address:	0A204h

Async Slice Count Select Register

DWord	Bit	Description
0	31:11	**Reserved** Default Value: 00000h Access: RO Reserved
	10:8	**Async_SS** Default Value: 010b Access: R/W Number of subslices to power (Async Mode): 001 : 1 subslice 010 : 2 subslices (GT1-based CHV, BSW only) Actual SSs used = Async_SS if SScountEn=0 Actual SSs used = SScount if SScountEn=1
	7:4	**Async_EU** Default Value: 1000b Access: R/W Maximum number of EUs to power per subslice if multiple subslices enabled Async Mode Actual EUs used = Async_EU if EUmin < Async_EU < EUmax Actual EUs used = EUmax if Async_EU >= EUmax Actual EUs used = EUmin if EUmin >= Async_EU
	3	**Reserved** Default Value: 0b Access: RO Reserved
	2:0	**temp_slicecount** Default Value: 000b Access: R/W Slice Count Request in Asynchronous Mode.

Auto Draw End Offset

3DPRIM_END_OFFSET - Auto Draw End Offset	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02420h-02423h
Valid Projects:	

DWord	Bit	Description
0	31:0	**End Offset** Format: U32 This register is used to store the end offset value used by the Vertex Fetch to determine when to stop processing the 3D_PRIMITIVE command. This register is valid when the End Offset Enable is set in the 3D_PRIMITIVE command.

Base of Data Stolen Memory

BDSM - Base of Data Stolen Memory

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0005Ch

This register contains the base address of Graphics Data Stolen DRAM memory.
Graphics Stolen Memory is within DRAM space. The base of stolen memory will always be below 4G.

DWord	Bit	Description
0	31:20	**BDSM** Default Value: 000h Access: R/W Lock BDSM: BASE_OF_Data_STOLEN_MEMORY. This register contains bits 31 to 20 of the base address of Data stolen DRAM memory. For certain GTLC generated accesses, this base register will be added to GTLC-provided offset address, forming the full physical address for the PFI fabric. This is also used as a base for VGA paged accesses.
	19:1	**RESERVED** Default Value: 00000h Access: RO Reserved
	0	**BDSM_LOCK** Default Value: 0b Access: R/W Lock This bit will lock all writable settings in this register, including itself.

Base of GTT Stolen Memory

BGSM - Base of GTT Stolen Memory

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00070h

Base of GTT table in Gfx Stolen Memory
The GTT table is located within Graphics Stolen Memory in DRAM space. The base of stolen memory will always be below 4G.

DWord	Bit	Description
0	31:20	**BGSM** Default Value: 000h Access: R/W Lock BGSM: Gfx Base of GTT Stolen Memory. This register contains bits 31 to 20 of the base address of GTT Table in stolen DRAM memory. BIOS determines base of GTT stolen memory by subtracting the GTT graphics stolen memory size (PCI offset 50 bits 9:8) from the Graphics Base of Data stolen(PCI offset 5C bits 31:20).
	19:1	**RESERVED** Default Value: 00000h Access: RO Reserved
	0	**BGSM_LOCK** Default Value: 0b Access: R/W Lock This bit will lock all writeable settings in this register including itself

Batch Address Difference Register

BB_ADDR_DIFF - Batch Address Difference Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02154h-02157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_RCSUNIT

Address:	12154h-12157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VCSUNIT0

Address:	1A154h-1A157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VECSUNIT

Address:	1C154h-1C157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_VCSUNIT1

Address:	22154h-22157h
Name:	Batch Address Difference Register
ShortName:	BB_ADDR_DIFF_BCSUNIT

This register contains the difference between the start of the last batch and where the last initiated Batch Buffer is currently fetching commands.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Address Difference** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned difference between the starting address of the batch buffer and where the last initiated Batch Buffer is currently fetching commands.
	1:0	**Reserved** Format: MBZ

Batch Buffer Head Pointer Register

BB_ADDR - Batch Buffer Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32

Address:	02140h-02143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_RCSUNIT

Address:	12140h-12143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VCSUNIT0

Address:	1A140h-1A143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VECSUNIT

Address:	1C140h-1C143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_VCSUNIT1

Address:	22140h-22143h
Name:	Batch Buffer Head Pointer Register
ShortName:	BB_ADDR_BCSUNIT

This register contains the current DWord Graphics Memory Address of the last-initiated batch buffer.

Programming Notes

Programming Restriction: This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Head Pointer** Project: CHV, BSW Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit will be 0 and this field will be meaningless.
	1	**Reserved** Format: MBZ

BB_ADDR - Batch Buffer Head Pointer Register

	0	**Valid** Format: U1 **Value** / **Name** / **Description** 0h / Invalid **[Default]** / Batch buffer Invalid 1h / Valid / Batch buffer Valid

Batch Buffer Per Context Pointer

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	021C0h-021C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_RCSUNIT
Address:	121C0h-121C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VCSUNIT0
Address:	1A1C0h-1A1C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VECSUNIT
Address:	1C1C0h-1C1C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_VCSUNIT1
Address:	221C0h-221C3h
Name:	Batch Buffer Per Context Pointer
ShortName:	BB_PER_CTX_PTR_BCSUNIT

This register is used to program the batch buffer address to be executed between context restore and execution of ring/execution list if enabled. This will only get executed due to regular context save/restore and not during power restore. This register is part of the execution list context and will be executed per context. Only supported if execution list is enabled. There is no preempting workloads within the Per Context Batch Buffer.

Source
BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
RenderCS
RenderCS
RenderCS

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Workaround
Workaround: [Render CS Only][Execlist Mode of Scheduling]: SW must ensure arbitration is switched off while context restore is in progress for any given context. This is achieved by disabling arbitration by programming MI_ARB_ON_OFF to "Arbitration Disable" in RCS_INDIRECT_CTX buffer and by enabling back the arbitration by programming MI_ARB_ON_OFF to "Arbitration Enable" as the last command prior to MI_BATCH_END in the BB_PER_CTX_PTR buffer of every context submitted. Note that RCS_INDIRECT_CTX_OFFSET could be set to default value or any other legitimate value as per the programming notes of the register definition. Arbitration disable by programming MI_ARB_ON_OFF (Arbitration Disabled) in RCS_INDIRECT_CTX buffer. Arbitration enabled by programming MI_ARB_ON_OFF (Arbitration Enabled) as the last command prior to MI_BATCH_BUFFER_END in BB_PER_CTX_PTR buffer. Additional Note: This WA need not be applied when it is guaranteed for no preemption to occur during execution of GPGPU workload. Preemption of GPGPU workload can be avoided by Bracketing the GPGPU workload with MI_ARB_ON_OFF (Arbitration Disable) and MI_ARB_ON_OFF (Arbitration Enable) command. MI_ARB_ON_OFF is a privileged command and can only be programmed in ring buffer or in privileged batch buffer (batch buffer in GGTT memory). Pending execlist submitted must not trigger preemption of the ongoing GPGPU workload due to following reasons First context of the pending execlist submitted is not the same as the ongoing GPGPU context. Force restore bit set for the submitted pending execlist.
Workaround: To work around a known HW issue, SW must do the below Programming Sequence prior to programming MI_BATCH_BUFFER_END command in BB_PER_CTX_PTR. SW must ensure both MI_LOAD_REGISTER_REG and MI_BATCH_BUFFER_END commands mentioned in the below sequence are placed in the same cacheline of memory. 1. MI_LOAD_REGISTER_IMM: 0x00800000 à 0x20C0 2. MI_ATOMIC a. Set “CS STALL” (Dword0[17]) b. “Return Data Control” enabled (Dword0[16]) c. “ATOMIC OPCODE” set to LOAD operation (Dword0[15:8]= 0x4) d. “Memory Address” set to scratch space in GFX memory. e. “Operand1 Data Dword 0” must be programmed to 0x0080_0080 3. MI_LOAD_REGISTER_MEM a. Set “Async Mode Enable” (Dword0[21]) b. “Memory Address” set to same as in MI_ATOMIC command above. c. “Register Address” set to 0x20C0 4. MI_LOAD_REGISTER_REG: 0x215C à 0x215C 5. MI_BATCH_BUFFER_END // Note that there shouldn’t be any commands programmed between step4 & step5 and also these commands must be placed in the same cacheline of memory. **Additional Note:**

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

This workaround need not be applied when Resource Streamer (RS) is not enabled or when a Resource Streamer enabled context is guaranteed not to be preempted.

- Preemption of RS enabled workload can be avoided by
- Bracketing the RS enabled workload with MI_ARB_ON_OFF (Arbitration Disable) and MI_ARB_ON_OFF (Arbitration Enable) command. MI_ARB_ON_OFF is a privileged command and can only be programmed in ring buffer or in privileged batch buffer (batch buffer in GGTT memory).
- Pending execlist submitted must not trigger preemption of the ongoing RS enabled workload due to following reasons
- First context of the pending execlist submitted is not the same as the ongoing RS enabled context.
- Force restore bit set for the submitted pending execlist.

DWord	Bit	Description
0	31:12	**Batch Buffer Per Context Address** Format: U20 Pointer to the Context in memory to be executed as a batch.
	11:2	**Reserved** Format: MBZ
	1	**RS Enabled Batch Buffer Per Context** Format: U1 If set, the command stream will enable the RS to parse commands. **Programming Notes** This must be set when programming the resource streamer pool commands (3DSTATE_BINDING_TABLE_POOL_ALLOC, 3DSTATE_GATHER_POOL_ALLOC, and 3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC) in order for the pool alloc fields to be valid in both the render engine and resource streamer.
	0	**Batch Buffer Per Context Valid** Format: U1 If set, the command stream will execute the context from the **Batch Buffer Per Context Address** prior to the execution of actual submitted workloads.

Batch Buffer Start Head Pointer Register

BB_START_ADDR - Batch Buffer Start Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02150h
Address:	12150h-12153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VCSUNIT0
Address:	1A150h-1A153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VECSUNIT
Address:	1C150h-1C153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_VCSUNIT1
Address:	22150h-22153h
Name:	Batch Buffer Start Head Pointer Register
ShortName:	BB_START_ADDR_BCSUNIT

This register contains the address specified in the last MI_START_BATCH_BUFFER command.

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Batch Buffer Start Head Pointer** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Batch Buffer starting address.
	1	**Preempted Batch Buffer RS Control Stop Flag** Format: Flag This field specifies RS Control Stop Flag when a batch buffer is preempted. This is for HW internal use and should not be written by SW. This bit gets reset when RS_PREEMPTED field of RS_PREEMPT_STATUS is written Zero.

BB_START_ADDR - Batch Buffer Start Head Pointer Register		
		This bit is set by: • Ctx restore of this bit • MI_RS_CONTROL_STOP (except for the ctx restore command) This bit is cleared by: • MI_RS_CONTROL_START • Any Batch start except resubmitted RS batch • A batch end that doesn't include preemption • Ctx save Writing 0 to bit[0] of the RS STATUS register
	0	**Reserved** Format: MBZ

Batch Buffer Start Head Pointer Register for Upper DWord

BB_START_ADDR_UDW - Batch Buffer Start Head Pointer Register for Upper DWord

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02170h
Address:	12170h-12173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VCSUNIT0
Address:	1A170h-1A173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VECSUNIT
Address:	1C170h-1C173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_VCSUNIT1
Address:	22170h-22173h
Name:	Batch Buffer Start Upper Head Pointer Register
ShortName:	BB_START_ADDR_UDW_BCSUNIT

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space specified in the last MI_START_BATCH_BUFFER command.

Programming Notes

Programming Restriction: This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Start Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space for the last initiated Batch Buffer starting address.

Batch Buffer State Register

BB_STATE - Batch Buffer State Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Access:	RO
Size (in bits):	32
Address:	02110h-02113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_RCSUNIT
Address:	12110h-12113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VCSUNIT0
Address:	1A110h-1A113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VECSUNIT
Address:	1C110h-1C113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_VCSUNIT1
Address:	22110h-22113h
Name:	Batch Buffer State Register
ShortName:	BB_STATE_BCSUNIT

This register contains the attributes of the current batch buffer initiated from the Ring Buffer.

This register should not be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer. This register is saved and restored with context.

DWord	Bit	Description
0	31:8	**Reserved** Project: All Format: MBZ
	7	**Reserved** Project: All Source: BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS Format: MBZ

BB_STATE - Batch Buffer State Register

	7	**Resource Streamer Enable** Project: CHV, BSW Source: RenderCS Format: U1 When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer.
	6	**Reserved**
	6	**2nd Level Buffer Security Indicator** Project: CHV, BSW Source: VideoCS, VideoCS2 Exists If: //VCS, VCS2 If set, VCS is fetching 2nd level batch commands from a PPGTT address space. If clear, GGTT. If execlists are enabled and this is set, the batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT. Note: When execlists are enabled this field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START.

Value	Name	Description
0h	MIBUFFER_SECURE **[Default]**	Located in GGTT memory
1h	MIBUFFER_NONSECURE	Located in PPGTT memory

	6	**2nd Level Buffer Security Indicator** Project: CHV, BSW Source: BlitterCS, VideoEnhancementCS Exists If: //BCS, VECS Format: MI_2ndBufferSecurityType If set, VECS is fetching 2nd level batch commands from a PPGTT address space. If clear, GGTT. If execlists are enabled and this is set, the batch buffer is non-secure and cannot execute privileged commands nor access privileged (GGTT) memory. It will be accessed via the PPGTT. If clear, this batch buffer is secure and will be accessed via the GGTT.

Value	Name	Description
0h	MIBUFFER_SECURE **[Default]**	Located in GGTT memory
1h	MIBUFFER_NONSECURE	Located in PPGTT memory

Programming Notes
When execlists are enabled, this field reflects the effective security level and may not be the same as the Buffer Security Indicator written using MI_BATCH_BUFFER_START.

BB_STATE - Batch Buffer State Register

	5	**Address Space Indicator** Project: CHV, BSW Note: This field reflects the effective address space indicator security level and may not be the same as the Address Space Indicator written using MI_BATCH_BUFFER_START. **Value** / **Name** / **Description** 0h / GGTT **[Default]** / This Batch buffer is located in GGTT memory and is privileged 1h / PPGTT / This Batch buffer is located in PPGTT memory and is non-privileged.
	4	**Reserved**
	4	**Reserved** Project: All Source: BlitterCS Exists If: //BCS Format: MBZ
	3:0	**Reserved** Project: All Format: MBZ

Batch Buffer Upper Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	0216Ch-0216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_RCSUNIT

Address:	1216Ch-1216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VCSUNIT0

Address:	1A16Ch-1A16Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VECSUNIT

Address:	1C16Ch-1C16Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_VCSUNIT1

Address:	2216Ch-2216Fh
Name:	Batch Buffer Upper Head Pointer Preemption Register
ShortName:	BB_PREEMPT_ADDR_UDW_BCSUNIT

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted batch buffer. This register follows the same rules as the BB_PREEMPT_ADDR register.

Programming Notes

Programming Restriction: This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted batch buffer.

Batch Buffer Upper Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32

Address:	02168h-0216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_RCSUNIT

Address:	12168h-1216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VCSUNIT0

Address:	1A168h-1A16Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VECSUNIT

Address:	1C168h-1C16Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_VCSUNIT1

Address:	22168h-2216Bh
Name:	Batch Buffer Upper Head Pointer Register
ShortName:	BB_ADDR_UDW_BCSUNIT

This register contains the current Upper DWord of Graphics Memory Address of the last-initiated batch buffer.
Programming Restriction:
This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit in BB_ADDR will be 0 and this field is meaningless.

BCS_PREEMPTION_HINT

BCS_PREEMPTION_HINT - BCS_PREEMPTION_HINT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	224BCh

This register contains the Head pointer offset in to the Ring Buffer or the Dword aligned Graphics address in to the Batch Buffer corresponding to either MI_ARB_CHECK called Preemption Hint Address. When Preemption Hint Address is enabled, BCS will honor UHPTR only on parsing MI_ARB_CHK at Preemption Hint Address.

This register contains the Head pointer offset into the Ring Buffer or the Dword aligned Graphics address into the Batch Buffer corresponding to the below listed commands in execlist mode of operation

- MI_ARB_CHECK
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT

Programming Notes

Programming Restriction: This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHECK in command stream. Programmer has to ensure that BCS Preemption Hint register gets programmed before UHPTR is programmed and well before BCS crosses the corresponding execution point. Preemption hint for both RingBuffer and Batch Buffer can't be enabled simultaneously.

DWord	Bit	Description
0	31:2	**Preempted Hint Address** Format: U30 Format: GraphicsAddress[31:2] This field contains the Head offset in to the Ring Buffer when Preemption Hint is set to Ring Buffer and Dword aligned Graphics Address in to the batch buffer when Preemption Hint is set to Batch Buffer.
	1	**Batch Buffer Preemption Hint** Format: Enable

Value	Name	Description
0h	Disabled	Preemption hint is disabled in batch buffer.
1h	Enabled	Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Batch Buffer.

BCS_PREEMPTION_HINT - BCS_PREEMPTION_HINT

	0	**Ring Preemption Hint** Format: Enable **Value** / **Name** / **Description** 0h / Disable / Preemption hint is disabled in ring buffer. 1h / Enabled / Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Ring Buffer.

BCS_PREEMPTION_HINT_UDW

BCS_PREEMPTION_HINT_UDW - BCS_PREEMPTION_HINT_UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	224C8h

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the Batch Buffer corresponding to MI_ARB_CHECK command called Preemption Hint Address.

Programming Notes

Programming Restriction: This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHK in command stream.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Preempted Hint Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer when Preemption Hint is set to Batch Buffer. This field is not valid when Preemption Hint is set to Ring Buffer.

BCS Active Upper Head Pointer Register

BCS_ACTHD_UDW - BCS Active Upper Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	2205Ch

This register contains the Head "Pointer" (4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space) of the currently-active batch buffer.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Head Pointer Upper DWORD** Default Value: 0h Format: GraphicsAddress[47:32] 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space corresponding to the Head Pointer of the currently-active batch buffer.

BCS Context ID Preemption Hint

BCS_CTXID_PREEMPTION_HINT - BCS Context ID Preemption Hint

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	224CCh

This register contains the Context ID of a context in execlist mode of operation. In execlist mode of operation BCS_PREEMPTION_HINT registers are looked at by Blitter Command Streamer on executing a context having Context ID that matches with the contents of this register. This register contents are valid and looked at only in execlist mode of operation.

Programming Notes

This register should NEVER be programmed in functional mode, this must be used only in validation mode to achieve deterministic preemption behavior in execlist mode of operation.

DWord	Bit	Description
0	31:0	**Context ID Preemption Hint** Format: U32 If 0 this field has no effect. If nonzero it indicates the only context ID that can be preempted when execlists are enabled. A preemption attempt when the context ID of the currently executing ring context does not match this field will be ignored.

BCS Context Sizes

BCS_CXT_SIZE - BCS Context Sizes

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000A05 CHV, BSW
Access:	Read/32 bit Write Only
Size (in bits):	32
Address:	221A8h

DWord	Bit	Description
0	31:13	**Reserved** Project: All Format: MBZ
	12:8	**BCS Context Size** Project: All Format: U5 Value: Ah — Name: **[Default]** — Project: CHV, BSW
	7:5	**Reserved** Project: All Format: MBZ
	4:0	**Execlist Context Size** Project: CHV, BSW Format: U5 Value: 5h — Name: **[Default]** — Project: CHV, BSW

BCS Context Timestamp Count

BCS_CTX_TIMESTAMP - BCS Context Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	223A8h

This register provides a mechanism to obtain cumulative run time of a GPU context on HW. This register gets context save/restored on a context switch. SW must reset this register on very first submission of a context to HW, then afterwards gets context save/restored maintaining the cumulative run time of the corresponding context. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the PIPE_CONTROL command. See 3D Geometry Pipeline in the "3D and Media" volume. This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register gets reset on an engine reset.

This register is context save restore on a context switch.

DWord	Bit	Description
0	31:0	**Timestamp Value** Format: U32 This register increments for every 80 ns of time.

BCS Counter for the Blitter Engine

BCS_CNTR - BCS Counter for the Blitter Engine		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	BlitterCS	
Default Value:	0xFFFFFFFF	
Access:	R/W	
Size (in bits):	32	
Address:	22178h	
DWord	**Bit**	**Description**
0	31:0	**Count Value** Default Value: ffffffffh Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter.

BCS Error Identity Register

BCS_EIR - BCS Error Identity Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/WC
Size (in bits):	32
Address:	220B0h

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s) except for the unrecoverable bits described).).

DWord	Bit	Description
0	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Error Identity Bits** Project: All Format: Array of Error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. To clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

Value	Name	Description
0h	**[Default]**	
1h	Error occurred	Error occurred

Programming Notes

Writing a '1' to a set bit will cause that error condition to be cleared. However, the Instruction Error bit (Bit 0) cannot be cleared except by reset (i.e., it is a fatal error).

BCS Error Mask Register

BCS_EMR - BCS Error Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0xFFFFFFFF CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	220B4h

The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.

Undefined or reserved bits in the Hardware Detected Error Bit Table will always return a read value of '1'

DWord	Bit	Description
0	31:16	**Reserved** Default Value: FFFFh Project: CHV, BSW Format: Must Be One
	15:0	**Error Mask Bits** Project: All Format: Array of error condition mask bits See Table 1 5. Hardware-Detected Error Bits This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description
0000h	Not Masked	Will be reported in the EIR
FFFFh	Masked **[Default]**	Will not be reported in the EIR

BCS Error Status Register

BCS_ESR - BCS Error Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	220B8h

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

DWord	Bit	Description
0	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Error Status Bits** Project: All Format: Array of error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description
0h	**[Default]**	
1h	Error Condition Detected	Error Condition detected

BCS Execute Condition Code Register

BCS_EXCC - BCS Execute Condition Code Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1
Address:	22028h

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a 1, while instruction is discarded if the condition evaluates to a 0. Once excluded, a ring is enabled into arbitration when the selected condition evaluates to a 0. This register also contains control for the invalidation of indirect state pointers on context restore.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] These bits serves as a write enable for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15	**Reserved** Format: MBZ
	14	**Context Wait for V-blank on Pipe-C** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe C Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	13	**Context Wait for V-blank on Pipe-B** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe B Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	12	**Context Wait for V-blank on Pipe-A** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe A Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.

BCS_EXCC - BCS Execute Condition Code Register

	11:5	**Reserved** Format: MBZ
	4:0	**Reserved** Project: CHV, BSW Format: MBZ

BCS General Purpose Register

BCS_GPR - BCS General Purpose Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	22600h-2267Fh

This is a general purpose register bank of sixteen 64-bit registers, used as temporary storage by the MI_MATH command to do ALU operations.

Programming Notes

Any operation that initiates a read to register 0x2266C will return the value of 0x2260c register. This does not include context save or MI_MATH command operation.

DWord	Bit	Description
0	63:0	**Reserved** Format: MBZ

BCS Hardware Status Mask Register

BCS_HWSTAM - BCS Hardware Status Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	22098h

Access: RO for Reserved Control bits

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes

To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled). At most 1 bit can be unmasked at any given time.

DWord	Bit	Description
0	31:0	**Hardware Status Mask Register** Default Value: FFFFFFFFh Project: All Format: Array of Masks refer to Table 5-1 in Interrupt Control Register section for bit definitions

BCS IDLE Max Count

BCS_PWRCTX_MAXCNT - BCS IDLE Max Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000040 CHV, BSW
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	22054h

This register contains the time in 0.64us to wait before telling power management hardware the render pipe is IDLE

DWord	Bit	Description
0	31:20	**Reserved** Project: All Format: MBZ
	19:0	**Reserved**

BCS Idle Switch Delay

BCS_IDLEDLY - BCS Idle Switch Delay

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	2223Ch

The IDLEDLY register contains an Idle Delay field which specifies the minimum number of microseconds allowed for command streamer to wait before a context is switched out leading to IDLE state in Execlist mode, i.e following this context switch there is no active element available in HW to execute.
A default value of 0, means that by default, there is no restriction to wait on a context switch leading to IDLE. This register has no significance when execlists are not enabled.

DWord	Bit	Description
0	31:21	**Reserved** Project: All Format: MBZ
	20:0	**IDLE Delay** Project: All Format: U21 Minimum number of micro-seconds allowed

BCS Instruction Parser Mode Register

BCS_INSTPM - BCS Instruction Parser Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	220C0h

Desc

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:11	**Reserved** Project: All Format: MBZ
	10	**Reserved**
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Reserved** Project: All Format: MBZ
	6:5	**Reserved** Project: CHV, BSW Format: MBZ
	4:0	**Reserved** Project: All Format: MBZ

BCS Interrupt Mask Register

BCS_IMR - BCS Interrupt Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	220A8h

The IMR register is used by software to control which Interrupt Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. "Masked" bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** Project: All Format: Array of interrupt mask bits Refer to Table 5-1 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description
FFFF FFFFh	**[Default]**	
0h	Not Masked	Will be reported in the IIR
1h	Masked	Will not be reported in the IIR

BCS Mode Register for Software Interface

BCS_MI_MODE - BCS Mode Register for Software Interface

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000200 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	2209Ch-2209Fh

The MI_MODE register contains information that controls software interface aspects of the command parser.

DWord	Bit	Description
0	31:16	**Masks** A 1 in a bit in this field allows the modification of the corresponding bit in Bits 15:0
	15	**Suspend Flush** Project: All Mask: MMIO(0x209c)#31 0h – No Delay – HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well 1h – Delay Flush – Suspend flush is active
	14:12	**Reserved** Read/Write
	11	**Invalidate UHPTR enable** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
	10	**Atomic Read Return for MI_COPY_MEM_MEM** Project: CHV, BSW 0h – Disable **[Default]** – Hardware does a regular memory fence write to complete the write to the destination address before moving to the next instruction. 1h – Enable – Hardware does Atomic Move with Read Return to complete the write to the destination address before moving to the next instruction.

BCS_MI_MODE - BCS Mode Register for Software Interface

9	**Ring Idle (Read Only Status Bit)** *Writes to this bit are not allowed.* **Value** / **Name** 0 – Parser not idle 1 – Parser idle **[Default]**
8	**Stop Ring** Software must set this bit to force the Ring and Command Parser to Idle. Software must read a 1 in Ring Idle bit after setting this bit to ensure that the hardware is idle. *Software must clear this bit for Ring to resume normal operation.* **Value** / **Name** 0 – Normal Operation **[Default]** 1 – Parser is turned off
7:2	**Reserved** Read/Write
1	**Bypass Fence Write** If set, this bit will bypass all writes during flushes, independent of programming. This includes post-sync op bits, the implicit TLB invalidate write (set in GFX_MODE[13]), and sync flush fences. *Note this is only intended for work-arounds* **Value** / **Name** 0 – Normal Operation 1 – Bypass
0	**Reserved** Read/Write

BCS PPGTT Directory Cacheline Valid Register

BCS_PP_DCLV - BCS PPGTT Directory Cacheline Valid Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	22220h

Default Value = 0h

This register controls update of the on-chip PPGTT Directory Cache during a context restore. Bits that are set will trigger the load of the corresponding 16 directory entry group. This register is restored with context (prior to restoring the on-chip directory cache itself). This register is also restored when switching to a context whose LRCA matches the current CCID if the Force PD Restore bit is set in the context descriptor.

The context image of this register must be updated and maintained by SW; SW should not normally need to read this register.

This register can also effectively be used to limit the size of a processes' virtual address space. Any access by a process that requires a PD entry in a set that is not enabled in this register will cause a fatal error, and no fetch of the PD entry will be attempted.

DWord	Bit	Description
0	63:32	**Reserved** Format: MBZ
	31:0	**PPGTT Directory Cache Restore** Format: Enable[32] [1..32] 16 entries If set, the [1st..32nd] 16 entries of the directory cache are considered valid and will be brought in on context restore. If clear, these entries are considered invalid and fetch of these entries will not be attempted.

BCS Primary DMA Engine Fetch Upper Address

BCS_DMA_FADD_P_UDW - BCS Primary DMA Engine Fetch Upper Address

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	22060h

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the instruction being fetched by the Primary DMA engine. This register contents are valid only when Batch Buffer is active.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Current DMA Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer that the "Primary" instruction parser DMA engine is currently accessing (fetching). Note that this address will typically lead the Head offset (as instructions must be fetched before execution).

BCS Reported Timestamp Count

BCS_TIMESTAMP - BCS Reported Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	RO. This register is not set by the context restore.
Size (in bits):	64
Address:	22358h

This register provides an elapsed real-time value that can be used as a timestamp.This register is not reset by a graphics reset. It will maintain its value unless a full chipset reset is performed. Note: This timestamp register reflects the value of the PCU TSC. The PCU TSC counts 10ns increments; this timestamp reflects bits 38:3 of the TSC (i.e. 80ns granularity, rolling over every 1.5 hours).

DWord	Bit	Description
0	63:36	**Reserved** Project: All Format: MBZ
	35:0	**Timestamp Value** Project: All Format: U36 This register toggles every 80 ns. The upper 28 bits are zero.

BCS Reset Control Register

BCS_RESET_CTRL - BCS Reset Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	220D0h

This register is to be used to control soft reset.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:2	**Reserved** Format: MBZ
	1	**Ready for Reset** Format: U1 When set indicates blitter engine is ready for reset. This bit gets cleared on engine reset or when Soft Reset In progress is cleared.
	0	**Request Reset** Format: U1 When set, this bit indicates that SW wishes to reset the blitter engine. On seeing this bit set, Command Streamer will take appropriate action and set Ready For Reset status bit. This bit gets cleared on engine reset.

BCS Ring Buffer Next Context ID Register

BCS_RNCID - BCS Ring Buffer Next Context ID Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	22198h-2219Fh

This register contains the *next* ring context ID associated with the ring buffer.

Programming Notes

The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that the only time a context switch can occur is when MI_ARB_CHECK enables preemption or the current context runs dry (head pointer becomes equal to tail pointer).

DWord	Bit	Description
0	63:0	**Unnamed** See Context Descriptor for BCS

BCS Semaphore Polling Interval on Wait

BCS_SEMA_WAIT_POLL - BCS Semaphore Polling Interval on Wait

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	2224Ch

The SEMA_WAIT_POLL register contains Poll Interval field which specifies the minimum number of microseconds allowed for command streamer to wait before re-fetching the data from the address mentioned in the MI_SEMAPHORE_WAIT command on WAIT Mode set to POLL until the condition is satisfied while the context is not switched out.
When value of 0 is written the poll interval will be equal to the memory latency of the read completion.

DWord	Bit	Description
0	31:21	**Reserved** Project: All Format: MBZ
	20:0	**Poll Interval** Project: All Format: U21 Minimum number of micro-seconds allowed

BCS SW Control

BCS_SWCTRL - BCS SW Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1
Address:	22200h

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask
	15:4	**Reserved** Project: CHV, BSW Format: MBZ
	3:2	**Reserved** Project: CHV, BSW Format: MBZ
	1	**Tile Y Destination** Project: CHV, BSW Format: U1 Programming this bit makes the HW treat all destination surfaces as Tile Y. This bit over-rides the setting of the destination format in the packet provided to the blitter command streamer. SW is required to flush the HW before changing the polarity of this bit. This bit is part of the context save/restore.
	0	**Tile Y Source** Project: CHV, BSW Format: U1 Programming this bit makes the HW treat all source surfaces as Tile Y. This bit over-rides the setting of the source format in the packet provided to the blitter command streamer. SW is required to flush the HW before changing the polarity of this bit. This bit is part of the context save/restore.

BCS Watchdog Counter Threshold

BCS_CTR_THRSH - BCS Watchdog Counter Threshold

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00150000
Access:	R/W
Size (in bits):	32
Address:	2217Ch

DWord	Bit	Description
0	31:0	**Counter logic Threshold** Default Value: 00150000h Format: U32 This field specifies the threshold that the hardware checks against for the value of the blitter clock counter before generating an interrupt. The counter in hardware generates an interrupt when the threshold is reached, rolls over and starts counting again. The interrupt generated is the "Media Hang Notify" interrupt since this watchdog timer is intended primarily to remedy VLD hangs on the main pipeline.

Bitstream Output Bit Count for the last Syntax Element Report Register

MFC_BITSTREAM_SE_BITCOUNT_SLICE - Bitstream Output Bit Count for the last Syntax Element Report Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128D4h
Name:	VDBOX1
Valid Projects:	CHV, BSW

This register stores the count of number of bits in the bitstream for the last syntax element before padding. The bit count is before the byte-aligned alignment padding insertion, but includes the stop-one-bit. This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC Bitstream Syntax Element Bit Count** Total number of bits in the bitstream output before padding. This count is updated each time the internal counter is incremented.

Bitstream Output Byte Count Per Slice Report Register

MFC_BITSTREAM_BYTECOUNT_SLICE - Bitstream Output Byte Count Per Slice Report Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128D0h
Valid Projects:	CHV, BSW

This register stores the count of bytes of the bitstream output. This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC Bitstream Byte Count** Total number of bytes in the bitstream output from the encoder. This count is updated for every time the internal bitstream counter is incremented.

Bitstream Output Minimal Size Padding Count Report Register

MFC_AVC_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12814h
Name:	VDBOX1
Valid Projects:	CHV, BSW

This register stores the count in bytes of **minimal size padding insertion. It is primarily provided for statistical data gathering.** This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC AVC MinSize Padding Count** Total number of bytes in the bitstream output contributing to minimal size padding operation. This count is updated each time when the padding count is incremented.

Blitter Mode Register

BLT_MODE - Blitter Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	2229Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**Execlist Enable** Default Value: 0h Project: CHV, BSW Mask: MMIO#31 When set, software can utilize the execlist registers to load a context into hardware. When this bit is clear the Execlist mechanism cannot be used. The ring must be loaded via MMIO access. **Programming Notes** This bit is not intended to be changed dynamically. Changing the value of this bit while rendering is in progress will have UNDEFINED results. This bit should be changed only after a full reset and before submitting any commands to the device
	14	**Interrupt Steering Bit** Project: CHV, BSW Format: U1 When set, Command Streamer sends interrupt messages to the SHIM of the ON CHIP Micro Controller through message channel.When reset, Command Streamer sends the interrupt messages to Display Engine as config writes on GAM interface.
	13:10	**Reserved** Project: All Format: MBZ

BLT_MODE - Blitter Mode Register

9 Per-Process GTT Enable

Project:	All
Format:	Enable Per-Process GTT BS Mode Enable

Value	Name	Description
0h	PPGTT Disable **[Default]**	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.

Programming Notes

This bit is used for enabling PPGTT access in Ring Buffer mode of scheduling. Privilege field in context descriptor states the same in Execlist Mode of scheduling. This field should be set before programming PDP0/1/2/3 registers in order to set the PPGTT translation of memory access.

8 Reserved

Project:	All

7 64Bit Virtual Addressing Enable

Project:	CHV, BSW
Format:	Enable

Per-Process GTT Enable

Value	Name	Description
0h	64Bit Virtual Addressing Disable **[Default]**	When clear indicates GFX operating in 32bit Virtual Addressing for PPGTT based memory access.

Programming Notes

This bit is only valid when PPGTT is enabled in ring buffer mode of scheduling. Context Descriptor has a similar bit to control 64bit virtual addressing in execlist mode of scheduling. Irrespective of this field set or clear virtual addresses translated through GGTT are always 32Bit. This field should be programmed before enabling PPGTT access. When this field is not set or for GGTT virtual addresses, Graphics Address [47:32] field of any commands or register exercised by SW should be programmed to 0x0.

64 Bit Virtual Addressing is not supported on CHV, BSW and must not be set.

6:5 Reserved

Project:	CHV, BSW

4 Reserved

Project:	CHV, BSW

BLT_MODE - Blitter Mode Register

	3:1	**Reserved** Project: All Format: MBZ
	0	**Privilege Check Disable** Project: CHV, BSW Format: Enable This field when set, disables Privilege Violation checks on non-privileged batch buffers. When set Privileged commands are allowed to be executed from non-privileged batch buffers.

Blitter TLB Control Register

BTCR - Blitter TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0426Ch

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

BLT Context Element Descriptor (High Part)

BLT_CTX_EDR_H - BLT Context Element Descriptor (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04504h

DWord	Bit	Description
0	31:0	**BLT Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W

BLT Context Element Descriptor (Low Part)

BLT_CTX_EDR_L - BLT Context Element Descriptor (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000009	
Size (in bits):	32	
Address:	04500h	
DWord	**Bit**	**Description**
0	31:0	**BLT Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W

BLT Context Element Descriptor (Low Part)

BLT_CTX_EDR_L - BLT Context Element Descriptor (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04500h

DWord	Bit	Description
0	31:0	**BLT Context Element Descriptor** Default Value: 00000009h Access: R/W

BLT PDP0/PML4/PASID Descriptor (High Part)

BLT_CTX_PDP0_H - BLT PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0450Ch

DWord	Bit	Description
0	31:0	**BLT PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W

BLT PDP0/PML4/PASID Descriptor (Low Part)

BLT_CTX_PDP0_L - BLT PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04508h

DWord	Bit	Description
0	31:0	**BLT PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W

BLT PDP1 Descriptor Register (High Part)

BLT_CTX_PDP1_H - BLT PDP1 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04514h

DWord	Bit	Description
0	31:0	**BLT PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP1 Descriptor Register (Low Part)

BLT_CTX_PDP1_L - BLT PDP1 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04510h

DWord	Bit	Description
0	31:0	**BLT PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

BLT PDP2 Descriptor Register (High Part)

BLT_CTX_PDP2_H - BLT PDP2 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0451Ch

DWord	Bit	Description
0	31:0	**BLT PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP2 Descriptor Register (Low Part)

BLT_CTX_PDP2_L - BLT PDP2 Descriptor Register (Low Part)			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	04518h		
DWord	**Bit**	**Description**	
0	31:0	**BLT PDP2 Descriptor Register (Low Part)**	
		Default Value:	00000000h
		Access:	R/W

BLT PDP3 Descriptor Register (High Part)

BLT_CTX_PDP3_H - BLT PDP3 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04524h

DWord	Bit	Description
0	31:0	**BLT PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

BLT PDP3 Descriptor Register (Low Part)

BLT_CTX_PDP3_L - BLT PDP3 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04520h

DWord	Bit	Description
0	31:0	**BLT PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

Boolean_Counter_B0

OAPERF_B0 - Boolean_Counter_B0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02920h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B0 to be read. Since what conditions cause B0 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B1

OAPERF_B1 - Boolean_Counter_B1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02924h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B1 to be read. Since what conditions cause B1 to increment are defined by the programming of CEC1-0/CEC1-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B2

OAPERF_B2 - Boolean_Counter_B2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02928h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B2 to be read. Since what conditions cause B2 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B3

OAPERF_B3 - Boolean_Counter_B3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0292Ch
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B3 to be read. Since what conditions cause B3 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B4

OAPERF_B4 - Boolean_Counter_B4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02930h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B4 to be read. Since what conditions cause B4 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B5

OAPERF_B5 - Boolean_Counter_B5	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02934h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B5 to be read. Since what conditions cause B5 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B6

OAPERF_B6 - Boolean_Counter_B6	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02938h
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B6 to be read. Since what conditions cause B6 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Boolean_Counter_B7

OAPERF_B7 - Boolean_Counter_B7	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BSpec
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0293Ch
Valid Projects:	[CHV, BSW]

This register enables the current live value of performance counter B7 to be read. Since what conditions cause B7 to increment are defined by the programming of CEC0-0/CEC0-1, please refer to the description of these registers and the values they hold to determine what event is reported via this register.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the live performance counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

BTB Not Consumed By RCS

BTP_PRODUCE_COUNT - BTB Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02480h

This register keeps track of the outstanding BTP produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

Programming Notes

This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**BTP Produce Count** This register keeps track of the outstanding BTP produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

BTP Commands Parsed By RCS

BTP_PARSE_COUNT - BTP Commands Parsed By RCS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02490h

This register keeps track of the BTP commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has BTP Pool Alloc Valid. BTP parse count should be less then equal to the BTP produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. BTP Parse count is subtracted from the BTP Produce Count upon parsing 3D_PRIMITIVE command. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**BTP Parse Count** This register keeps track of the BTP commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has BTP Pool Alloc Valid. BTP parse count should be less then equal to the BTP produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. BTP Parse count is subtracted from the BTP Produce Count upon parsing 3D_PRIMITIVE command.

Cache Mode Register 0

CACHE_MODE_0 - Cache Mode Register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000004 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	07000h
Valid Projects:	CHV, BSW

Description

This register is used to control the operation of the Render and Sampler L2 Caches. All reserved bits are implemented as read/write.
Before changing the value of this register, GFX pipeline must be idle i.e. full flush is required.
This Register is saved and restored as part of Context.

RegisterType = MMIO_SVL

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask[15:0] A 1 in a bit in this field allows the modification of the corresponding bit in Bits 15:0.
	15	**Sampler L2 Disable** Project: CHV, BSW Access: r/w Format: Disable 0h **[Default]** – Sampler L2 Cache Enabled. 1h – Sampler L2 Cache Disabled. All accesses are treated as misses.
	14:12	**MSAA Compression Plane Number Threshold for eLLC** Project: CHV, BSW Security: IP.eLLC Access: r/w 0h threshold0 **[Default]** – Cache only planeID = 0 in eLLC. 1h threshold1 – Cache only planeID = 0, 1 in eLLC.

CACHE_MODE_0 - Cache Mode Register 0

2h	threshold2	Cache only planeID = 0..2 in eLLC.
3h	threshold3	Cache only planeID = 0..3 in eLLC.
4h	threshold4	Cache only planeID = 0..4 in eLLC.
5h	threshold5	Cache only planeID = 0..5 in eLLC.
6h	threshold6	Cache only planeID = 0..6 in eLLC.
7h	threshold7	Cache only planeID = 0..7 in eLLC.

Programming Notes
This bit-field is programmed based on MSAA. When MSAA compression is enabled, these settings affect HW, else it is ignored. For 16X MSAA only lower 8 planes can be cached in eLLC.

11 **Sampler Set Remmapping for 3D Disable**

Project:	CHV, BSW
Access:	r/w

Value	Name	Description
0h	Enable Set Remap **[Default]**	Set remapping for 3d enabled
1h	Disable Set Remap	Set remapping for 3d disabled

10 **Reserved**

Project:	CHV, BSW
Format:	Disable

This bit controls the bug-fix in the allocation pipe for expansions when the PMA-optimization mode is enabled.

Value	Name	Description
0h	Enable **[Default]**	Allocation pipe is not stalled if there are pending expansions
1h	Disable	Allocation pipe is stalled if there are pending expansions

9 **Sampler L2 TLB Prefetch Enable**

Access:	r/w

Value	Name	Description
0h	**[Default]**	TLB Prefetch Disabled
1h		TLB Prefetch Enabled

8 **Reserved**

CACHE_MODE_0 - Cache Mode Register 0

7:6 **Sampler L2 Request Arbitration**

Project:	CHV, BSW
Access:	r/w
Format:	U2

Value	Name	Description
00b		Round Robin
01b		Fetch are Highest Priority
10b		Constants are Highest Priority
11b		Reserved

5 **STC Eviction Policy**

Project:	CHV, BSW
Access:	r/w
Format:	Disable

If this bit is set, STCunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.

Programming Notes	Project
If this bit is set to "1", bit 4 of 0x7010h must also be set to "1".	CHV, BSW

4 **RCC Eviction Policy**

Project:	CHV, BSW
Access:	r/w
Format:	Disable

If this bit is set, RCCunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates that non-LRA eviction policy. This bit must be reset. LRA replacement policy is not supported.

Programming Notes	Project
If this bit is set to "1", bit 7 of 0x7010h must also be set to "1".	CHV, BSW

3 **Reserved**

CACHE_MODE_0 - Cache Mode Register 0

2 **Hierarchical Z RAW Stall Optimization Disable**

Project:	CHV, BSW
Access:	r/w
Format:	U1

The Hierarchical Z RAW Stall Optimization allows non-overlapping polygons in the same 8x4 pixel/sample area to be processed without stalling waiting for the earlier ones to write to Hierarchical Z buffer.

Value	Name	Description
0h	Enable	Enables the hierarchical Z RAW Stall Optimization.
1h	Disable **[Default]**	Disables the hierarchical Z RAW Stall Optimization.

Programming Notes	Project
This bit must be set to 0 to enable the Hierarchical Z RAW stall optimization.	CHV, BSW

1 **Disable clock gating in the pixel backend**

Access:	r/w
Format:	Disable

MCL related clock gating is disabled in the pixel backend. Before setting this bit to 1, the instruction/state caches must be invalidated. [DevGT:{WKA}]

0 **RW optimization (portal2) fix disable**

Project:	CHV, BSW
Access:	r/w

Portal 2 fix :
a) In MSAA mode, RCC source clear cacheline will be allocated as RW, only if its a true read (ie blend enabled, etc) and will be allocated as WO otherwise.
b) In any mode, RCC should not allocate a cacheline as RW, if its not a true read, evenif MSC sends RW = 11 to RCC. MSC sends an additional "True read' indicator to help RCC override the RW bit.

Value	Name	Description
0	**[Default]**	Portal 2 fix enabled.
1		Portal 2 fix disabled. a) RCC will always allocate source clear as RW in MSAA mode. b) RCC will always allocate on the basic of MSC RW indicator to RCC, in non source clear mode.

Cache Mode Register 1

CACHE_MODE_1 - Cache Mode Register 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000180 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	07004h
Valid Projects:	CHV, BSW

Description
RegisterType: MMIO_SVL
Before changing the value of this register, GFX pipeline must be idle; i.e., full flush is required. This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Mask: MASK Format: Mask[15:0] Must be set to modify corresponding data bit. Reads to this field returns zero.
	15	**MSC Smart Bank Arbitration Disable** Project: CHV, BSW Access: r/w Format: Disable Setting this bit causes MSC Bank Arbitration to be disabled. Default value, i.e. resetting this bit, will Enable MSC Smart Bank Arbitration. 0h – Enable **[Default]** 1h – Disable

CACHE_MODE_1 - Cache Mode Register 1

14 **MSC Resolve Optimization Disable**

Project:	CHV, BSW
Access:	r/w
Format:	U1

Setting this bit causes MSC to mark cachelines dirty and appropriately update MSC during the classic clear resolve pass. Default value, i.e. resetting this bit, suppresses MSC buffer modification during the classic clear resolve pass.

Value	Name
1h	Disable
0h	Enable **[Default]**

13 **NP EARLY Z FAILS DISABLE**

Project:	CHV, BSW
Access:	r/w

Value	Name	Description
0h	Disable **[Default]**	When NP PMA FIX ENABLE = 1, clearing this bit disables IZ to conservatively fail pixels.
1h	Enable	When NP PMA FIX ENABLE = 1, IZ does conservatively fail any NP pixels.

Programming Notes
This bit must be set when NP PMA FIX ENABLE = 1
This bit must not be set when NP PMA FIX ENABLE = 0

12 **HIZ Eviction Policy**

Project:	CHV, BSW
Access:	r/w
Format:	U1

If this bit is set, Hizunit will have LRA as replacement policy. The default value i.e. (when this bit is reset) indicates the non-LRA eviction policy. For performance reasons, this bit must be reset.

Value	Name	Description
0h	**[Default]**	Non-LRA eviction Policy
1h		LRA eviction Policy

Programming Notes
If this bit is set to "1", bit 3 of 0x7010h must also be set to "1"

CACHE_MODE_1 - Cache Mode Register 1

11 **NP PMA FIX ENABLE**

Project:	CHV, BSW
Access:	r/w

Value	Name	Description
0h	Disable **[Default]**	Enables stalling PMA behavior for NP depth pixels in the early depth pipeline. (Legacy behavior)
1h	Enable	Enables non-stalling PMA behavior for NP depth pixels in the early depth pipeline.

Programming Notes

SW must set this bit in order to enable this fix when following expression is TRUE. 3DSTATE_WM::ForceThreadDispatch != 1 && !(3DSTATE_RASTER::ForceSampleCount != NUMRASTSAMPLES_0) && (3DSTATE_DEPTH_BUFFER::SURFACE_TYPE != NULL) && (3DSTATE_DEPTH_BUFFER:: HIZ Enable) && !(3DSTATE_WM::EDSC_Mode == 2) && (3DSTATE_PS_EXTRA::PixelShaderValid) && !(3DSTATE_WM_HZ_OP::DepthBufferClear || 3DSTATE_WM_HZ_OP::DepthBufferResolve || 3DSTATE_WM_HZ_OP::Hierarchical Depth Buffer Resolve Enable || 3DSTATE_WM_HZ_OP::StencilBufferClear) && (3DSTATE_WM_DEPTH_STENCIL::DepthTestEnable) && (((3DSTATE_PS_EXTRA::PixelShaderKillsPixels || 3DSTATE_PS_EXTRA:: oMask Present to RenderTarget || 3DSTATE_PS_BLEND::AlphaToCoverageEnable || 3DSTATE_PS_BLEND::AlphaTestEnable || 3DSTATE_WM_CHROMAKEY::ChromaKeyKillEnable) && (3DSTATE_WM::ForceKillPix != ForceOff && ((3DSTATE_WM_DEPTH_STENCIL::DepthWriteEnable && 3DSTATE_DEPTH_BUFFER::DEPTH_WRITE_ENABLE) || (3DSTATE_WM_DEPTH_STENCIL::Stencil Buffer Write Enable && 3DSTATE_DEPTH_BUFFER::STENCIL_WRITE_ENABLE && 3DSTATE_STENCIL_BUFFER::STENCIL_BUFFER_ENABLE)))) || (3DSTATE_PS_EXTRA:: Pixel Shader Computed Depth mode != PSCDEPTH_OFF))

10 **Reserved**

Project:	CHV, BSW

9 **Reserved**

Project:	CHV, BSW
Access:	r/w

CACHE_MODE_1 - Cache Mode Register 1

8:7 **Sampler Cache Set XOR selection**

Project:	CHV, BSW
Access:	r/w
Format:	U2

These bits have an impact only when the Sampler cache is configured in 16 way set associative mode. If the cache is being used for immediate data or for blitter data these bits have no effect.

Value	Name	Description
00b	None	No XOR.
01b	Scheme 1	New_set_mask[3:0] = Tiled_address[16:13]. New_set[3:0] less than or = New_set_mask[3:0] ^Old_set[3:0]. Rationale: These bits can distinguish among 16 different equivalent classes of virtual pages. These bits also represent the lsb for tile rows ranging from a pitch of 1 tile to 16 tiles.
10b	Scheme 2	New_set_mask[3] = Tiled_address[17] ^ Tiled_address[16].New_set_mask[2] = Tiled_address[16] ^ Tiled_address[15].New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14].New_set_mask[0] = Tiled_address[14] ^ Tiled_address[13].New_set[3:0] less than or = New_set_mask[3:0] ^ Old_set[3:0].Rationale: More bits on each XOR can give better statistical uniformity on sets and since two lsbs are taken for each tile row size, it reduces the chance of aliasing on sets.
11b	Scheme 3 **[Default]**	New_set_mask[3] = Tiled_address[22] ^ Tiled_address[21] ^ Tiled_address[20] ^ Tiled_address[19].New_set_mask[2] = Tiled_address[18] ^ Tiled_address[17] ^ Tiled_address[16].New_set_mask[1] = Tiled_address[15] ^ Tiled_address[14].New_set_mask[0] = Tiled_address[13].New_set[3:0] less than or = New_set_mask[3:0] ^ Old_set[3:0].Rationale: More bits on each XOR can give better statistical uniformity on sets and since each XOR has different bits, it reduces the chance of aliasing on sets even more.

Programming Notes

This field should be programmed as “00b” corresponding to NO XOR option when the 3D map performance fix in MT is enabled using the field “**Sampler Set Remmapping for 3D Disable”** in **CACHE_MODE_0 - Cache Mode Register 0.**

CACHE_MODE_1 - Cache Mode Register 1

6 **4X4 RCPFE-STC Optimization Disable**

Project:	CHV, BSW
Access:	r/w
Format:	Disable

Value	Name	Description
0h	**[Default]**	Enables two contiguous 4x2s to be collected as 4X4 access for STC interface. This allows for less bank collision and less RAM power on STC.
1h		Disables this optimization and therefore only one valid 4x2 is sent to STC on the 4X4 interface.

5 **MCS Cache Disable**

Project:	CHV, BSW
Access:	r/w
Format:	Disable

For Programming restrictions please refer to the 3D Pipeline.

Value	Name	Description
0h	**[Default]**	MCS cache enabled. It allows RTs with MCS buffer enabled to be rendered using either MSAA compression for MSRT OR with color clear feature for non MSRT.
1h		MCS cache is disabled. Hence no MSAA compression for MSRT and no color clear for non-MSRT.

4 **Float Blend Optimization Enable**

Project:	CHV, BSW
Access:	r/w
Format:	Enable

Value	Name	Description
0h	**[Default]**	Disables blend optimization for floating point RTs.
1h		Enables blend optimization for floating point RTs.

3 **Depth Read Hit Write-Only Optimization Disable**

Project:	CHV, BSW
Access:	r/w
Format:	Disable

Value	Name	Description
0h	**[Default]**	Read Hit Write-only optimization is enabled in the Depth cache (RCZ).
1h		Read Hit Write-only optimization is disabled in the Depth cache (RCZ).

CACHE_MODE_1 - Cache Mode Register 1

2 **RCZ Read after expansion control fix 2**

Access:	r/w
Format:	Enable

Value	Name	Description
0h	**[Default]**	RCZ will suppress the read request to memory if it was allocated as a expansion Cacheline RCZ will always issue a read request to memory, even if it was previously allocated as expansion Cacheline
1h		RCZ will always issue a read request to memory, even if it was previously allocated as expansion Cacheline RCZ will suppress the read request to memory if it was allocated as a expansion Cacheline

1 **Blend Optimization Fix Disable**

Project:	CHV, BSW
Access:	r/w

Description
This bit when reset, enables blend optimization fix. If this bit is set, it disables the blend optimization fix and may exhibit corruption on alpha components in the render target under some conditions.

Value	Name
0	**[Default]**
1	

0 **Reserved**

Project:	CHV, BSW

CAPPOINT

CAPPOINT - CAPPOINT

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x000000D0
Size (in bits):	32
Address:	00034h

This register points to a linked list of capabilities implemented by this device.

DWord	Bit	Description
0	31:8	**RESERVED** Default Value: 000000h Access: RO Reserved
	7:0	**CAPABILITIES_POINTER** Default Value: D0h Access: R/W Once The first item in the capabilities list is at address D0h (PMCS). This register is programmed by BIOS during boot-up.Once written, this register becomes Read_Only. This register can only be cleared by a Reset.

CCK_CTL1

CCK_CTL1 - CCK_CTL1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00200000
Size (in bits):	32
Address:	101100h

BGF related programming.Note: This register should be the last one written between 10_1100, 10_1104, 10_1108. To be more specific, 10_1100[0] is the last one to be programmed between 10_1100h, 10_1104h and 10_1108h registers. The reason is that CZ clock does not stop on a frequency change and hence the fused values for the CZ clock domain must be the same and not intermittently transition to an invalid value, like all zeros.Changed for CHV, BSW.

DWord	Bit	Description
0	31:28	**czcount_30ns** Default Value: 0h Access: R/W **Description** This register contains the number of cz clock cycles that it takes to make up 30ns.

Frequency (MHz)	czcount_30ns
400	4'b1011
333.333	4'b1001
320	4'b0001
266.667	4'b0111
200	4'b0101

Frequency (MHz)	czcount_30ns
533.333	4'b1111
466.667	4'b1101
373.333	4'b1011
360	4'b1010
355.556	4'b1010
350	4'b1010

DWord	Bit	Description
	27	**Reserved** Default Value: 0b Access: RO Reserved

CCK_CTL1 - CCK_CTL1

26:24 **WellPUfreq**

Default Value:	000b
Access:	R/W

While Render or Media is being waking up from standby, clocks are allowed to be run for some time for x-contention removal. For CHV, BSW, there is a perceived di/dt risk by running this x-contention removal at full CU2X/CU frequency. This potential issue is amplified for CHV, BSW supports a 1:1 primary gear ratio and having a CU2X clock frequency of upto 2G.
* 0xx: x-contention removal time will use: CU2X/1 for 2x clock, CU/1 for 1X clock.
* 100: x-contention removal time will use a CU/4.
* 101: x-contention removal time will use a CU/8.
* 110: x-contention removal time will use a CU/16.
* 111: x-contention removal time will use a CU/32.

Programming Notes

Punit should never program this bitfield to 3'b000 and it should program this to 3'b110.

23 **CFG1Fuse0**

Default Value:	0b
Access:	R/W

bgfsource_config1fuse0. By default, there is no frequency change. The set of BGF and SS registers are provided by fuses. When frequency change is required, this bit must be set to '1' and Punit/driver must program the remainder of the clock crossing registers.
This bit must be last bit written between register 10_1100, 10_1104 and 10_1108.
The following registers are affected by this configuration bit: selecting between fuse values and configuration register:
0x10_1100[26:24]
0x10_1100[22:16]
0x10_1104[31:0]
0x10_1108[31:0]

22:20 **CZ2CU_PTRSEP**

Default Value:	010b
Access:	R/W

bgf_cz2cu_ptrsep. Pointer separation for cz2cu BGF
000 = Pointer separation of 0
001 = Pointer separation of 1
010 = Pointer separation of 2 (Primary)
011 = Pointer separation of 3 (Secondary)
100 = Pointer separation of 4
101 = Pointer separation of 5
110 = Pointer separation of 6
111 = Pointer separation of 7

CCK_CTL1 - CCK_CTL1

	19	**Reserved** Default Value: 0b Access: RO Reserved
	18:16	**CU2CZ_PTRSEP** Default Value: 0h Access: R/W bgf_cu2cz_ptrsep. Pointer separation for cu2cz BGF 000 = Pointer separation of 0 001 = Pointer separation of 1 010 = Pointer separation of 2 (Primary) 011 = Pointer separation of 3 (Secondary) 100 = Pointer separation of 4 101 = Pointer separation of 5 110 = Pointer separation of 6 111 = Pointer separation of 7
	15:9	**Reserved** Default Value: 00h Access: RO Reserved
	8	**ClkSquashEn** Default Value: 0b Access: R/W Clock Squash Enable. 0 = disabled. 1 = enabled. When this bit is enabled the GFX clock squashing feature is enabled. Clock squashing is only allowed when VNN is set to Vmin. **Programming Notes** This field must be zero at all times when in HPLLmode.
	7:1	**Reserved** Default Value: 0h Access: RO Reserved
	0	**DblBufPulse** Default Value: 0h Access: WO Reading from this register will always return a zero. Punit will write a one into this bit in order to push the BGF parameters into the double buffer during CPD/clock squah being enabled. A write to this register bit creates a one clock wide pulse (doublebufferpulse)to the CPunit.

CCK_CTL2

CCK_CTL2 - CCK_CTL2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	101104h

CZ and CU ratios for BGFs and for common clock calculations.Changed for CHV, BSW.

DWord	Bit	Description
0	31:16	**CZ2CU_CUratio** Default Value: 0000h Access: R/W These 16 bits shows the CU part of the BGF ratio between CZ and CU clock domain. CU_SquashAllow (=CZ2CU_CUratio) must always be less than or equal to CU_SquashWindow. When ClkSquashEn=0, CU_SquashAllow = CZ2CU_CUratio = CU_SquashWindow. 10_1108h will be dont care when ClkSquashEn=0. The effective/average squashed frequency will be determined by:(CU_SquashAllow/CU_SquashWindow) When ClkSquashEn=1 and in GPLLmode, 12 <= CZ2CU_CUratio(10_1104[31:16]) <= CU_Squashwindow(10_1108[31:16]) and multiples of 4. Effective CUfrequency = CUratio/(CU Squash Window) Actual CUfrequency = UnsquashedFrequency*CUratio/(CU Squash Window) If in GPLLmode, CU2x frequency = (400/10)*N, where N inside {2,50} (333/8)*N, where N inside {2,48} (320/8)*N, where N inside {2,50} (267/6)*N, where N inside {2,45} (200/5)*N, where N inside {2,50} The value programmed into this bitfield will be N*4. In HPLLmode: CZ2CU_CUratio=720/(INTEGER*2). The HPLLvco (as determined by CCK_FUSE_REG0[1:0]) is divided by an INTEGER=(1,2,3,4,5,6,8,9,10) to create the CU2x frequency. CUclk=CU2xclk/2.

CCK_CTL2 - CCK_CTL2

	15:0	**CZ2CU_CZratio** Default Value: 0000h Access: R/W These 16 bits shows the CU part of the BGF ratio between CZ and CU clock domain. CZ2CU_CZratio = UsyncPeriod/CZperiod In GPLLmode, UsyncPeriod is expected to be (8*GPLLref_in_ns). GPLLref_in_ns is expected to be CZperiod*5 for CZ200 CZperiod*6 for CZ266 CZperiod*8 for CZ320 CZperiod*8 for CZ333 CZperiod*10 for CZ400. That makes CZ2CU_CZratio = 8*5 for CZ200 and GPLL mode. CZ2CU_CZratio = 8*6 for CZ266 and GPLL mode. CZ2CU_CZratio = 8*8 for CZ320 and GPLL mode. CZ2CU_CZratio = 8*8 for CZ333 and GPLL mode. CZ2CU_CZratio = 8*10 for CZ400 and GPLL mode. The above only applies for GPLL mode and when UsyncPeriod = 8*GPLLref, CPunit will internally generate the following signal: CZ_Com_limit = CZ2CU_CZratio - 1 In HPLLmode: CZ2CU_CZratio=720/(INTEGER*1). The HPLLvco (as determined by CCK_FUSE_REG0[1:0]) is divided by an INTEGER=(1,2,3,4,5,6,8,9,10) to create the CZ frequency.

CCK_CTL3

CCK_CTL3 - CCK_CTL3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00008000
Size (in bits):	32
Address:	101108h

Graphics clock squash control.Changed for CHV, BSW.

DWord	Bit	Description
0	31:16	**CU SQUASH WINDOW** Default Value: 0000h Access: R/W This register will always show the number of unsquashed clock cycles in a usync window. CU_SquashWindow = UsyncPeriod/Unsquashed_CUperiod. In GPLLmode, UsyncPeriod is expected to be (8*GPLLref_in_ns). CUperiod will be GPLLref_in_ns/Ratio. That makes, CU_SquashWindow = (8*GPLLref_in_ns)/(GPLLref_in_ns/Ratio) = 8*Ratio. Ratio is the multiplier from GPLLref to provide the CU clock frequency. Note: CU_SquashWindow = 8*Ratio only applies for GPLL mode when the usync period is 8*GPLL. In HPLLmode, clock squashing is not supported and 10_1100[8] must be zero at all times in HPLLmode.
	15:0	**CU SQUASH INIT** Default Value: 8000h Access: R/W This register is used to move around the location of where the clocks are squashed. For initial design, this is always set to 8000h.

CCK_CTL4

CCK_CTL4 - CCK_CTL4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	10110Ch

CZclk 30ns TSV Generation

DCN requires support for more CZclk frequencies

DWord	Bit	Description
0	31:17	**SPARE** Default Value: 00000000h Access: R/W Reserved
	16	**CZCLK 30ns Mechanism** Access: R/W This bit defines whether we use the old counter mechanism for generating the 30ns pulse or use this new TSV scheme 0 - Use the old counter mechanism to determine the 30ns pulse (default) 1 - Use the new TSV scheme with the Allow and Window values from this register
	15:8	**CZclk HPLL VCOS IN 30ns** Access: R/W These bits determine the window in terms of the number of HPLL VCO clock cycles in 30ns
	7:0	**CZclk HPLL DIV** Access: R/W These bits define the integer used to divide HPLLvco to get CZclk

CLAIM_ER

CLAIM_ER - CLAIM_ER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182028h

Claim Error Counter - counts the RM claim error (no RM claim) reported at CLAIM_ERROR field of EIR

DWord	Bit	Description
0	31	**CLAIM_ER_CLR** Default Value: 0b Access: WO writing a '1' to this bit clears the CLAIM_ER_CTR, CTR_OVERFLOW and CLAIM_ERROR bit (EIR[0]). A read of this bit always returns 0
	30:17	**RESERVED** Default Value: 0000h Access: RO Reserved
	16	**CTR_OVERFLOW** Default Value: 0b Access: RO When set, indicates that the CLAIM_ER_CTR field of this register has overflowed (i.e., claim-error count has reached the value of 65536) The overflow is supplied by the RMbus master.
	15:0	**CLAIM_ER_CTR** Default Value: 0000h Access: RO

Clipper Invocation Counter

CL_INVOCATION_COUNT - Clipper Invocation Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02338h
Valid Projects:	

This register stores the count of objects entering the Clipper stage. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**CL Invocation Count Report UDW** Number of objects entering the clipper stage. Updated only when Statistics Enable is set in CLIP_STATE (see the Clipper Chapter in the 3D Volume.)
	31:0	**CL Invocation Count Report LDW** Number of objects entering the clipper stage. Updated only when Statistics Enable is set in CLIP_STATE (see the Clipper Chapter in the 3D Volume.)

Clipper Primitives Counter

CL_PRIMITIVES_COUNT - Clipper Primitives Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02340h
Valid Projects:	

This register reflects the total number of primitives that have been output by the clipper. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**Clipped Primitives Output Count UDW** Total number of primitives output by the clipper stage. This count is updated for every primitive output by the clipper stage, as long as Statistics Enable is set in SF_STATE (see the Clipper and SF Chapters in the 3D Volume.)
	31:0	**Clipped Primitives Output Count LDW** Total number of primitives output by the clipper stage. This count is updated for every primitive output by the clipper stage, as long as Statistics Enable is set in SF_STATE (see the Clipper and SF Chapters in the 3D Volume.)

CLKGATE Messaging Register for Clocking Unit

MSG_CLKGATE_GCP - CLKGATE Messaging Register for Clocking Unit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	0802Ch

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:7	**Reserved** Access: RO
	6	**Gate cmclk Acknowledgement (VCS1)** Access: R/W Gate cmclk Acknowledgement (VCS1) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated
	5	**Gate cwclk Acknowledgement (WIN)** Access: R/W Gate cwclk Acknowledgement (WIN) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated
	4	**Reserved**
	3	**Gate cfclk Acknowledgement (CS)** Access: R/W Gate cfclk Acknowledgement (CS) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated
	2	**Gate cvclk Acknowledgement (VECS)** Access: R/W Gate cvclk Acknowledgement (VECS) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated

MSG_CLKGATE_GCP - CLKGATE Messaging Register for Clocking Unit

	1	**Gate cmclk Acknowledgement (VCS0)** Access: R/W Gate cmclk Acknowledgement (VCS0) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated
	0	**Gate crclk Acknowledgement (CS)** Access: R/W Gate crclk Acknowledgement (CS) 1'b0 : Clocks are ungated <default> 1'b1 : Clocks are gated

CLOCK_GATE_DIS1

CLOCK_GATE_DIS1 - CLOCK_GATE_DIS1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182060h

Gunit Clock Gating Disable

DWord	Bit	Description
0	31	**GIOSF_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	30	**GPFI_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	29	**GRMW_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	28	**GSA_OUTBOUND_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	27	**GSA_RTN_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating

CLOCK_GATE_DIS1 - CLOCK_GATE_DIS1

	26	**GUP_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	25	**GWAKE_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	24	**GCR_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	23	**GBC_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	22	**GINT_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	21	**GRMBUS_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	20	**GCCBCZ_KEYCMP_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating

CLOCK_GATE_DIS1 - CLOCK_GATE_DIS1

	19:16	**SPARE1** Default Value: 0h Access: R/W Spare
	15	**Reserved**
	14	**Reserved**
	13	**Reserved**
	12	**GSECDISP_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	11	**GREGDISP_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	10	**GCTLQ_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	9	**GMMIO_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	8	**GCFG_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	7:4	**SPARE2** Default Value: 0h Access: R/W Spare

CLOCK_GATE_DIS1 - CLOCK_GATE_DIS1		
	3:0	**CZ_IDLE_TIMER** Default Value: 0h Access: R/W Gunit CZ domain idle timer count value. Individual Gunit CZ fubs use this as a counter pre-load value. 0 = Idle timer disabled 1 - 15 = After a unit indicates idle, wait this many clocks before gating.

CLOCK_GATE_DIS2

CLOCK_GATE_DIS2 - CLOCK_GATE_DIS2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182064h

Gunit Clock Gating Disable register2.

DWord	Bit	Description
0	31:9	**CLKGATE_SPARE** Default Value: 000000h Access: R/W Spare to be assigned to clock gate disables.
	8	**GPMH_DIS** Default Value: 0b Access: R/W New for CHV, BSW 0 = Enable clock gating 1 = Disable clock gating
	7	**ATOMICS_DIS** Default Value: 0b Access: R/W New for CHV, BSW 0 = Enable clock gating 1 = Disable clock gating
	6	**IOSFSB_P_DIS** Default Value: 0b Access: R/W New for CHV, BSW 0 = Enable clock gating 1 = Disable clock gating
	5	**IOSFSB_DIS** Default Value: 0b Access: R/W New for CHV, BSW 0 = Enable clock gating 1 = Disable clock gating

CLOCK_GATE_DIS2 - CLOCK_GATE_DIS2

	4	**GARB_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	3	**GMAP_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	2	**Reserved**
	1	**GRDDATBUF_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating
	0	**GWRDATQ_DIS** Default Value: 0b Access: R/W 0 = Enable clock gating 1 = Disable clock gating

Clock gate and clear EU metrics

EUMETRICSCLEAR - Clock gate and clear EU metrics

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138198h

EU Metrics Clear

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000h Access: RO Reserved
	0	**Clear EU Metrics** Default Value: 0b Access: R/W For Cdyn optimization, when this bit is set to one, all of the 13 counters are clock gated. Then, after setting it to '0(zero), On the falling edge of EUMetricsClr, all of the 13 counters are reset to zero and counting resumes.

Event Name	Units	Maximum per cycle (average)	Metric	Comments
event0: ss0_gpm_sampler_active_igpa	1 = 1 sampler busy in this sub-slice.	1 sampler busy in this sub-slice.	Sampler busy per sub-slice	Measures sampler utilization (i.e, sampler not idle)
event1: ss1_gpm_sampler_active_igpa	1 = 1 sampler busy in this sub-slice.	1 sampler busy in this sub-slice.	Sampler busy per sub-slice	Measures sampler utilization (i.e, sampler not idle)
event2: tdl0_gpm_eu_stall	1 = 32 EUs stalled on this sub-slice.	8 EUs stalled on this sub-slice.	EU stall per sub-slice	Measures EU stalls
event3: tdl1_gpm_eu_stall	1 = 32 EUs stalled on this sub-slice.	8 EUs stalled on this sub-slice.	EU stall per sub-slice	Measures EU stalls

EUMETRICSCLEAR - Clock gate and clear EU metrics

		event4: tdl0_gpm_ipc_count	1 = 32 instructions executed on this sub-slice.	16 instructions executed on this sub-slice.	IPC per sub-slice	Measures EU utilization: 0, 1, or 2 counts per clock per EU
		event5: tdl1_gpm_ipc_count	1 = 32 instructions executed on this sub-slice.	16 instructions executed on this sub-slice.	IPC per sub-slice	Measures EU utilization: 0, 1, or 2 counts per clock per EU
		event6: tdl0_gpm_threads_allocated	1 = 64 threads allocated in this sub-slice.	56 threads allocated in this sub-slice.	Thread allocated per sub-slice	Measures the threads allocated
		event7: tdl1_gpm_threads_allocated	1 = 64 threads allocated in this sub-slice.	56 threads allocated in this sub-slice.	Thread allocated per sub-slice	Measures the threads allocated
		event8: tdl0_gpm_eu_stall_sampler	1 = 64 EUs stalled by sampler in this sub-slice.	56 threads stalled by sampler in this sub-slice.	Placeholder for EUstallTime by sampler per sub-slice	Measures how much of eu stalls are due to sampler
		event9: tdl1_gpm_eu_stall_sampler	1 = 64 EUs stalled by sampler in this sub-slice.	56 threads stalled by sampler in this sub-slice.	Placeholder for EUstallTime by sampler per sub-slice	Measures how much of eu stalls are due to sampler
		event10: tdl0_gpm_eu_not_idle	1 = 32 EUs not idle in this sub-slice.	8 EUs not idle in this sub-slice.	Clocks any EU is not idle per sub-slice	Measures EU utilization (i.e, EU not idle)
		event11: tdl1_gpm_eu_not_idle	1 = 32 EUs not idle in this sub-slice.	8 EUs not idle in this sub-slice.	Clocks any EU is not idle per sub-slice	Measures EU utilization (i.e, EU not idle)
		event12: constant_one	1			one clock cycle toggles.

Clock Gating Messages

CGMSG - Clock Gating Messages

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	08104h

Clock Gating Messages Register

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:7	**Reserved** Access: RO Reserved
	6	**Media 1 Clock gating control message** Access: R/W Gate Media 1 (2nd Vbox) Clock Message : '0' : Media 1 Clock Un-gate Request (un-gates the cmclk clock in the 2n Media block) '1' : Media 1 Clock Gate Request (gates the cmclk clockin the 2nd Media block)
	5	**WIDI CLock Gating control Message** Access: R/W Gate WIDI Clock Message : '0' : WIDI Clock Un-gate Request (un-gates the cwclk clock) '1' : WIDI Clock Gate Request (gates the cwclk clock)
	4	**Reserved**
	3	**Fix Function Clock gating Control Message** Access: R/W Gate Fix Clock Message : '0' : Fix Clock Un-gate Request (un-gates the cfclk/cf2xclk clock) '1' : Fix Clock Gate Request (gates the cfclk/cf2xclk clock)
	2	**VEbox Clock gating Control message** Access: R/W Gate VE-box Clock Message : '0' : VEbox Clock Un-gate Request (un-gates the cvclk clock) '1' : VEbox Clock Gate Request (gates the cvclk clock)

CGMSG - Clock Gating Messages		
	1	**Media 0 Clock Gating Control Message** Access: R/W Gate Media Clock Message : '0' : Media 0 Clock Un-gate Request (un-gates the cmclk clock) '1' : Media 0 Clock Gate Request (gates the cmclk clock)
	0	**Row Clock Gating Control Message** Access: R/W Gate Row Clocks Message : '0' : Row Clock Un-gate Request (un-gates the crclk and cr2xclk clocks) '1' : Row Clock Gate Request (gates the crclk and cr2xclk clocks)

Color/Depth Write FIFO Watermarks

CZWMRK - Color/Depth Write FIFO Watermarks

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04060h

This register is directly mapped to the current Virtual Addresses in the MTTLB (Texture and constant cache TLB).

DWord	Bit	Description
0	31:24	**Reserved** Format: MBZ
	23:18	**Color Wr Burst Size** This is the maximum size of the requests burst, from the last High Watermark trip, before reevaluating the High Watermark again.
	17:16	**Reserved** Format: MBZ
	15:12	**Color Wr FIFO High Watermark** This is the number of accumulated Color writes that will trigger a Burst of Z Writes.
	11:6	**Z Wr Burst Size** This is the maximum size of the requests burst, from the last High Watermark trip, before reevaluating the High Watermark again.
	5:4	**Reserved** Format: MBZ
	3:0	**Z Wr FIFO High Watermark** This is the number of accumulated Depth writes that will trigger a Burst of Z Writes.

Configuration Register0 for RPMunit

CONFIG0 - Configuration Register0 for RPMunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x00000000
Size (in bits):	32
Address:	00D00h

Lock bit LOCK applies to all RW/L fields in this register. Lock is overridden during context restore.

DWord	Bit	Description
0	31	**Lock for RW/L Fields in this Register** Access: R/W Lock 0 = Bits of CONFIG0 register are R/W. 1 = All bits of CONFIG0 register are RO (including this lock bit). Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). Lock is reset on a restore after context is captured.
	30:0	**Placeholder Bits** Project: CHV, BSW Access: R/W Lock Placeholder bits for implementation or ECO loops.

Configuration Register1 for RPMunit

CONFIG1 - Configuration Register1 for RPMunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x00000000
Size (in bits):	32
Address:	00D04h

Lock bit LOCK applies to all RW/L fields in this register. Lock is overridden during context restore.

DWord	Bit	Description
0	31	**Lock for RW/L Fields in this Register** Access: R/W Lock 0 = Bits of CONFIG0 register are R/W. 1 = All bits of CONFIG0 register are RO (including this lock bit). Once written to 1, the lock is set and cannot be cleared (i.e., writing a 0 will not clear the lock). Lock is reset on a restore after context is captured.
	30:0	**Placeholder Bits** Project: CHV, BSW Access: R/W Lock Placeholder bits for implementation or ECO loops.

Configuration Register for RCPunit

RCPCONFIG - Configuration Register for RCPunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x0000000F
Size (in bits):	32
Address:	00D08h

Unit Level Clock Gating Control Registers

DWord	Bit	Description
0	31:5	**Placeholder Bits** Access: R/W Lock Placeholder bits for implementation or ECO loops.
	4	**Reserved**
	3	**RPMunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).
	2	**MGSRunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).
	1	**MDRBunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).

RCPCONFIG - Configuration Register for RCPunit

	0	**MCRunit Clock Gating Disable in Uncore Well** Default Value: 1b Access: R/W Clock Gating Disable Control: '0': Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality). '1': Clock Gating Disabled. (i.e., clocks are toggling, always) (DEFAULT).

Context Load Protocol Register BLT

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	04014h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - BCS 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - BCS 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - BCS 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - BCS 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - BCS 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT

10	**Context Load Protocol Register - BCS 10**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
9	**Context Load Protocol Register - BCS 9**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
8	**Context Load Protocol Register - BCS 8**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
7	**Context Load Protocol Register - BCS 7**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
6	**Context Load Protocol Register - BCS 6**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
5	**Context Load Protocol Register - BCS 5**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
4	**Context Load Protocol Register - BCS 4**	
	Default Value:	0b
	Access:	R/W
	For Future Use. This bit is self clear.	
3	**Context Load Protocol Register - BCS 3**	
	Default Value:	0b
	Project:	CHV, BSW
	Access:	R/W

BLT_CTX_LD_PRTCL - Context Load Protocol Register BLT

		Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - BCS 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 2 = Request from BCS to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - BCS 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - BCS 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by BCS) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register CS

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000 CHV, BSW	
Size (in bits):	32	
Address:	04004h	

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - CS 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - CS 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - CS 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - CS 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - CS 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS

	10	**Context Load Protocol Register - CS 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - CS 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - CS 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - CS 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - CS 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - CS 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - CS 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**Context Load Protocol Register - CS 3** Default Value: 0b Project: CHV, BSW Access: R/W

GFX_CTX_LD_PRTCL - Context Load Protocol Register CS

		Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - CS 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 2 = Request from CS to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - CS 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - CS 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by CS) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register VCS0

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	04008h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VCS0 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VCS0 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VCS0 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VCS0 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VCS0 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0

	10	**Context Load Protocol Register - VCS0 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VCS0 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VCS0 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VCS0 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VCS0 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VCS0 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VCS0 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

MFX0_CTX_LD_PRTCL - Context Load Protocol Register VCS0

	3	**Context Load Protocol Register - VCS0 3** Default Value: 0b Project: CHV, BSW Access: R/W Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - VCS0 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS0) Bit 2 = Request from VCS0 to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - VCS0 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS0) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - VCS0 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS0) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register VCS1

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	0400Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VCS1 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VCS1 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VCS1 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VCS1 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VCS1 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

	10	**Context Load Protocol Register - VCS1 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VCS1 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VCS1 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VCS1 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VCS1 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VCS1 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VCS1 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

MFX1_CTX_LD_PRTCL - Context Load Protocol Register VCS1

	3	**Context Load Protocol Register - VCS1 3** Default Value: 0b Project: CHV, BSW Access: R/W Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - VCS1 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS1) Bit 2 = Request from VCS1 to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - VCS1 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS1) Bit 1 = Context Launched This bit is self clear.
	0	**Context Load Protocol Register - VCS1 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VCS1) Bit 0 = Context Available. This bit is self clear.

Context Load Protocol Register VEBX

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	04010h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Context Load Protocol Register - VEBX 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**Context Load Protocol Register - VEBX 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**Context Load Protocol Register - VEBX 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**Context Load Protocol Register - VEBX 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**Context Load Protocol Register - VEBX 11** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX

	10	**Context Load Protocol Register - VEBX 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**Context Load Protocol Register - VEBX 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**Context Load Protocol Register - VEBX 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**Context Load Protocol Register - VEBX 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**Context Load Protocol Register - VEBX 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**Context Load Protocol Register - VEBX 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**Context Load Protocol Register - VEBX 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

VEBX_CTX_LD_PRTCL - Context Load Protocol Register VEBX

	3	**Context Load Protocol Register - VEBX 3** Default Value: 0b Project: CHV, BSW Access: R/W Bit 3 = Context Complete. Only valid with Bit 2 asserted and indicates whether the context switching out as complete vs. incomplete. This bit is self clear.
	2	**Context Load Protocol Register - VEBX 2** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VEBX) Bit 2 = Request from VEBX to GAM for context save readiness. GAM will acknowledge appropriated steps taken. This bit is self clear.
	1	**Context Load Protocol Register - VEBX 1** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VEBX) Bit 1 = Context Launched. This bit is self clear.
	0	**Context Load Protocol Register - VEBX 0** Default Value: 0b Access: R/W Context Load Protocol Register (Written by VEBX) Bit 0 = Context Available. This bit is self clear.

Context Restore Request To TDL

TDL_CONTEXT_RESTORE - Context Restore Request To TDL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E440h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:17	**Reserved** Format: MBZ
	16	**Context Restore Mask** Value: 1 — Name: — Description: Bit 0 and bit 16 both need to be 1 for Context restore request
	15:1	**Reserved** Format: MBZ
	0	**Context Restore** Value: 1 — Name: — Description: Bit 0 and bit 16 both need to be 1 for Context restore request

Context Save Request To TDL

TDL_CONTEXT_SAVE - Context Save Request To TDL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4FCh
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:17	**Reserved** Format: MBZ
	16	**Context Save Mask** Value: 1 — Name: — Description: Bit 0 and Bit 16 both need to be '1' for Context Save Request
	15:1	**Reserved** Format: MBZ
	0	**Context Save** Value: 1 — Name: — Description: Bit 0 and Bit 16 both need to be '1' for Context Save Request

Context Sizes

CXT_SIZE - Context Sizes

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x05655582
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	021A8h

The actual size of a logical rendering context is the amount of data stored/restored during a context switch and is measured in 64B cache lines.

This register will be power context save/restored. Note that this register will default to the correct value, so software should not have to modify it.

DWord	Bit	Description
0	31:27	**Reserved** Format: MBZ
	26:24	**Ring Context Size** Default Value: 5h This field indicates the Ring context data that needs to be save/restored.
	23:16	**Render Context Size** Default Value: 65h This field indicates the size of the render context data that needs to be save/restored when extended mode is not enabled for a context; this also excludes VF, VFE, and URB context size.
	15:8	**SOL Context Offset** Default Value: 55h This field indicates the cacheline aligned offset of the SOL context in the render context image starting from Ring Context. Note that in execlist of scheduling Ring context itself is at 4KB offset from LRCA.
	7:0	**VF and VFE State Context Size** Default Value: 82h This field indicates the amount of VF and VFE unit data context save/restored in cachelines.

Context Status Buffer Contents

CTXT_ST_BUF - Context Status Buffer Contents

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000
Access:	R/W
Size (in bits):	384
Trusted Type:	1
Address:	02370h-0239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_RCSUNIT
Address:	12370h-1239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VCSUNIT0
Address:	1A370h-1A39Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VECSUNIT
Address:	1C370h-1C39Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_VCSUNIT1
Address:	22370h-2239Fh
Name:	Context Status Buffer Contents
ShortName:	CTXT_ST_BUF_BCSUNIT

Contents of the Execlist 0 in HW.

Programming Notes

This structure contains the Context Switch status locations Context Status 0 to Context Status 5.

DWord	Bit	Description
0	63:32	**Context Status 0 UDW** Format: Context Status CHV, BSW
	31:0	**Context Status 0 LDW** Format: Context Status CHV, BSW
1	63:32	**Context Status 1 UDW** Format: Context Status CHV, BSW

CTXT_ST_BUF - Context Status Buffer Contents		
	31:0	**Context Status 1 LDW** Format: Context Status CHV, BSW
2	63:32	**Context Status 2 UDW** Format: Context Status CHV, BSW
	31:0	**Context Status 2 LDW** Format: Context Status CHV, BSW
3	63:32	**Context Status 3 UDW** Format: Context Status CHV, BSW
	31:0	**Context Status 3 LDW** Format: Context Status CHV, BSW
4	63:32	**Context Status 4 UDW** Format: Context Status CHV, BSW
	31:0	**Context Status 4 LDW** Format: Context Status CHV, BSW
5	63:32	**Context Status 5 UDW** Format: Context Status CHV, BSW
	31:0	**Context Status 5 LDW** Format: Context Status CHV, BSW

Control Register for Power Management

WAIT_FOR_RC6_EXIT - Control Register for Power Management	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	020CCh
Address:	120CCh-120CFh
Name:	Control Register for Power Management
ShortName:	WAIT_FOR_RC6_EXIT_VCSUNIT0
Address:	1A0CCh-1A0CFh
Name:	Control Register for Power Management
ShortName:	WAIT_FOR_RC6_EXIT_VECSUNIT
Address:	1C0CCh-1C0CFh
Name:	Control Register for Power Management
ShortName:	WAIT_FOR_RC6_EXIT_VCSUNIT1
Address:	220CCh-220CFh
Name:	Control Register for Power Management
ShortName:	WAIT_FOR_RC6_EXIT_BCSUNIT

DWord	Bit	Description
	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:14	**Reserved** Format: MBZ

WAIT_FOR_RC6_EXIT - Control Register for Power Management

0

This register gets power context save/restored. Bit[0] contents of this register doesn't get save/restored.

13 **Selective Read Addressing Enable**

Project:	CHV, BSW

This field controls the outbound read request originating from Render Command Streamer. This field enables to read the MMIO register from selected unit in a given slice and sub-slice instead of multicasting the read cycle to all slices/sub-slices.

Value	Name	Description
0h	**[Default]**	Lowest Slice and Lowest Sub-Slice Enabled. Ex: Slice-0, Sub-Slice-0 are the lowest in GT.
1h	Selective Unit Enabled	Unit selected based on **Selective Read Slice Select** and **Selective Read Sub-Slice Select**.

12:11 **Selective Read Slice Select**

Project:	CHV, BSW

This field selects the slice from which the read return data value has to be considered when **Selective Read Addressing Enable** is set. Below value must be programmed with a legal value supported by the GT configuration, one must not program this to a slice that is disabled or not supported by GT.

Value	Name
00b	Slice-0
01b	Slice-1
10b	Slice-2
11b	Reserved

10:9 **Selective Read Sub-Slice Select**

Project:	CHV, BSW

This field selects the sub-slice from which the read return data value has to be considered when **Selective Read Addressing Enable** is set. Below value must be programmed with a legal value supported by the GT configuration, one must not program this to a sub-slice that is disabled or not supported by GT.

Value	Name
00b	Sub Slice-0
01b	Sub Slice-1
10b	Sub Slice-2
11b	Reserved

WAIT_FOR_RC6_EXIT - Control Register for Power Management

8 **Render Inhibit**

Format:	Disable

Value	Name	Description
0h	Disabled **[Default]**	When not Set CS doesn't take any special action.
1h	Enabled	When Set CS will not save/restore render context as part of power context save/restore. Render context includes RS context as well if enabled.

Programming Notes
If this bit is set S/W should set Resource Streamer Context Enable (Bit[7] of this register)as well.

7 **Resource Streamer Context Enable**

Format:	Enable

Value	Name	Description
1h	Disable	The current context does not include the resource streamer context
0h	Enable **[Default]**	The current context does include the resource streamer context.

6 **Selective Write Addressing Enable**

Project:	CHV, BSW

This field controls the outbound write request on message channel originating from Render Command Streamer on executing MI_LOAD_REGISTER_IMM, MI_LOAD_REGISTER_REG and MI_LOAD_REGISTER_MEM commands. Setting this field doesn't affect the execution of MI_LOAD_REGISTER_IMM command from context image during context restore. This field enables to direct the message channel write cycle to the unit in the selected slice and sub-slice instead of multicasting it to all the instances of the unit in all the slices and sub-slices.

Value	Name	Description
0h	Multi Cast **[Default]**	
1h	Selective Unit Enabled	Unit selected based on **Selective Write Slice Select** and **Selective Write Sub-Slice Select**.

WAIT_FOR_RC6_EXIT - Control Register for Power Management

5:4 Selective Write Slice Select

Project:	CHV, BSW

This field selects the slice to which the write has to be done when **Selective Write Addressing Enable** is set. Below value must be programmed with a legal value supported by the GT configuration, one must not program this to a slice that is disabled or not supported by GT.

Value	Name
00b	Slice-0
01b	Slice-1
10b	Slice-2
11b	Reserved

3:2 Selective Write Sub-Slice Select

Project:	CHV, BSW

This field selects the Sub-Slice to which the write has to be done when **Selective Write Addressing Enable** is set. Below value must be programmed with a legal value supported by the GT configuration, one must not program this to a sub-slice that is disabled or not supported by GT.

Value	Name
00b	Sub Slice-0
01b	Sub Slice-1
10b	Sub Slice-2
11b	Reserved

1 Reserved

Format:	MBZ

0 WAIT FOR RC6 EXIT

Format:	Disable

Value	Name	Description
0h	Disabled **[Default]**	When not Set CS doesn't take any action.
1h	Enabled	When Set CS will stop on the next appropriate command boundary and will initiate IDLE sequence with PM.

Programming Notes
WAIT_FOR_RC6_EXIT functionality is only supported in ring buffer mode of scheduling and not supported in execlist mode of scheduling.

Count Active Channels Dispatched

TS_GPGPU_THREADS_DISPATCHED - Count Active Channels Dispatched

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02290h

This register is used to count the number of active channels that TS sends for dispatch. For each dispatch the active bits in the execution mask are summed and added to this register. This register is reset when a write occurs to 2290h

DWord	Bit	Description
0	63:32	**GPGPU_THREADS_DISPATCHED UDW** Format: U32 This count is increased by the number of active bits in the execution mask each time the TS sends a GPGPU dispatch.
	31:0	**GPGPU_THREADS_DISPATCHED LDW** Format: U32 This count is increased by the number of active bits in the execution mask each time the TS sends a GPGPU dispatch.

CS Context Timestamp Count

CS_CTX_TIMESTAMP - CS Context Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	023A8h

This register provides a mechanism to obtain cumulative run time of a GPU context on HW. This register gets context save/restored on a context switch. SW must reset this register on very first submission of a context to HW, then afterwards gets context save/restored maintaining the cumulative run time of the corresponding context. This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register gets reset on an engine reset.

This register is context save restore on a context switch. The time to execute the context switch is included in the CS_CTX_TIMESTAMP register.

DWord	Bit	Description
0	31:0	**Timestamp Value** Format: U32 This register increments for every 80 ns of time.

CS General Purpose Register

CS_GPR - CS General Purpose Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02600h
Name:	CS General Purpose Register 0
ShortName:	CS_GPR_R_0
Address:	02608h
Name:	CS General Purpose Register 1
ShortName:	CS_GPR_R_1
Address:	02610h
Name:	CS General Purpose Register 2
ShortName:	CS_GPR_R_2
Address:	02618h
Name:	CS General Purpose Register 3
ShortName:	CS_GPR_R_3
Address:	02620h
Name:	CS General Purpose Register 4
ShortName:	CS_GPR_R_4
Address:	02628h
Name:	CS General Purpose Register 5
ShortName:	CS_GPR_R_5
Address:	02630h
Name:	CS General Purpose Register 6
ShortName:	CS_GPR_R_6
Address:	02638h
Name:	CS General Purpose Register 7
ShortName:	CS_GPR_R_7
Address:	02640h
Name:	CS General Purpose Register 8
ShortName:	CS_GPR_R_8

CS_GPR - CS General Purpose Register	
Address:	02648h
Name:	CS General Purpose Register 9
ShortName:	CS_GPR_R_9
Address:	02650h
Name:	CS General Purpose Register 10
ShortName:	CS_GPR_R_10
Address:	02658h
Name:	CS General Purpose Register 11
ShortName:	CS_GPR_R_11
Address:	02660h
Name:	CS General Purpose Register 12
ShortName:	CS_GPR_R_12
Address:	02668h
Name:	CS General Purpose Register 13
ShortName:	CS_GPR_R_13
Address:	02670h
Name:	CS General Purpose Register 14
ShortName:	CS_GPR_R_14
Address:	02678h
Name:	CS General Purpose Register 15
ShortName:	CS_GPR_R_15
Address:	12600h-12607h
Name:	General Purpose Register
ShortName:	CS_GPR_VCSUNIT0
Address:	1A600h-1A607h
Name:	General Purpose Register
ShortName:	CS_GPR_VECSUNIT
Address:	1C600h-1C607h
Name:	General Purpose Register
ShortName:	CS_GPR_VCSUNIT1
Address:	22600h-22607h
Name:	General Purpose Register
ShortName:	CS_GPR_BCSUNIT
This is a General Purpose Register bank of sixteen 64bit registers, which will be used as temporary storage by MI_MATH command to do ALU operations.	

CS_GPR - CS General Purpose Register

GPR Index	MMIO Offset
R_0	0x2600
R_1	0x2608
R_2	0x2610
R_3	0x2618
R_4	0x2620
R_5	0x2628
R_6	0x2630
R_7	0x2638
R_8	0x2640
R_9	0x2648
R_10	0x2650
R_11	0x2658
R_12	0x2660
R_13	0x2668
R_14	0x2670
R_15	0x2678

DWord	Bit	Description
0	63:0	**CS_GPR_DATA** Project: CHV, BSW This register is a temporary register for ALU operations. See MI_MATH command for more details.

CS Power Management FSM

CSPWRFSM - CS Power Management FSM

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	022ACh

This register contains the state code of the Power Management FSM, FBC Forward FSM, CSSTDT FSM, CSARB FSM, CSBUPDATE FSM. Decoding the contents of this register will indicate what the state of the corresponding state machine.

DWord	Bit	Description
0	31:30	**Reserved** Format: MBZ
	29:28	**CSFBCSLICE0** Format: U2 FBC message forward FSM state

Value	Name
0h	CSFBCIDLE_0
1h	CSFBCMODIFY_0
2h	CSFBCCLEAN_0
3h	CSFBCDONE_0

	27:24	**Reserved** Format: MBZ
	23:21	**CS ARB** Format: U3 Overall state of the command streamer. Describes what state CS is in

Value	Name	Project
0h	ARBIDLE_s	
1h	P0RNG_s CS	
2h	P0BATCH_s	
3h	ARBCHK	
4h	ARBCHK1	
5h	CTXOP_s	
6h	WABATCH_s	CHV, BSW
7h	PSLBATCH	CHV, BSW

CSPWRFSM - CS Power Management FSM

20	**Reserved** Format: MBZ
19:17	**CSSWITCH** Format: U3 Arbiters CSSWITCH FSM state decoding.

Value	Name
0h	SWIDLE_s
1h	SWITCH_s
2h	ASREQ_s
3h	DMACHK_s
4h	ARBWAIT_s
5h	FIFORECFG_s
6h-7h	Reserved

16:13 **CSCSBUPDATE**

Format: U4

CS Power Management CSBLOCK FSM state

Value	Name
0h	CSBIDLE
1h	CSQ
2h	WRPTR
3h	SEMA1
4h	SEMA2
5h	TS1
6h	TS2
7h	TS3
8h	TS4
9h	DUMMYREQ
Ah	DUMMYWT
Bh	INTWT
Ch-Fh	Reserved

CSPWRFSM - CS Power Management FSM

	12:11	**R2MWRREQ** Format: U2 CSSTDT memory request FSM state

Value	Name
0h	WRIDLE
1h	WRREQ_HW1
2h	WRREQ_HW2
3h	WRRD

	10	**Reserved** Format: MBZ
	9:7	**LOADARB** Format: U3 CSSTDT arbiter FSM state

Value	Name
0h	LDIDLE
1h	LDAUTO
2h	LDPRSR
3h	LDCTX
4h	LDFLSH
5h	LDREG
6h	LDSHR1

Programming Notes
LOADARB FSM states needs 4 bits for encoding, however only 3bits have been mapped on MMIO. 8h -> LDLRM is the state which is missed out, due to less bits mapped, LDLRM/LDIDLE cant be resolved with certain.

	6:4	**CSBLOCK** Format: U3 CS Power Management CSBLOCK FSM state

Value	Name
0h	CSBLOCK
1h	CSCTXARB
2h	CSUNBLOCKRESTORE
3h	CSUNBLOCK
4h	CSPREP4BLOCK
5h-7h	Reserved

CSPWRFSM - CS Power Management FSM

	3:0	**CSIDLE** Format: U4 CS Power Management CSBLOCK FSM state

Value	Name
0h	CSBUSY
1h	CNTWT
2h	FLSHREQ
3h	FLSHWT
4h	CTXSAVE
5h	CSREQBLOCK
6h	PMTURNOFF
7h	PMIDLEWT
8h	IDLE
9h	PMTURNON
Ah	PMBUSYWT
Bh	DOPFFCGREQ
Ch	DOPFFCGWAIT
Dh	DOPFFCG
Eh	DOPFFCUGREQ
Fh	DOPFFCUGWAIT

CSPREEMPT

CSPREEMPT - CSPREEMPT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	024B0h

Programming Notes

This is for HW internal usage and must not be written by SW.

DWord	Bit	Description
0	31:16	**Mask Bits** Project: CHV, BSW Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Unnamed** Project: CHV, BSW Format: Disable This is a message bit written by the cross CS in case of GT4-CBR/SFR mode of operation. To set this bit both bit[0] and bit[16] (mask) needs to be set. This bit set indicates CS in other GT has reached a preemption point. This bit gets reset by CS when preemption takes place.

CS Reset Control Register

CS_RESET_CTRL - CS Reset Control Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	020D0h
This register is to be used to control soft reset.	

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:2	**Reserved** Format: MBZ
	1	**Ready for Reset** Format: U1 When set indicates render engine is ready for reset. This bit gets cleared on engine reset or when Soft Reset In progress is cleared.
	0	**Request Reset** Format: U1 When set indicates SW wishes to reset the render engine. On seeing this bit set Command Streamer will take appropriate action and set Ready For Reset status bit. This bit gets cleared on engine reset.

Current Context Register

CCID - Current Context Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02180h
Address:	12180h-12183h
Name:	Current Context Register
ShortName:	CCID_VCSUNIT0
Address:	1A180h-1A183h
Name:	Current Context Register
ShortName:	CCID_VECSUNIT
Address:	1C180h-1C183h
Name:	Current Context Register
ShortName:	CCID_VCSUNIT1
Address:	22180h-22183h
Name:	Current Context Register
ShortName:	CCID_BCSUNIT

This register contains the current logical rendering context address associated with the ring buffer in ring buffer mode of scheduling. This register contents are not valid in Exec-List mode of scheduling.

Programming Notes

The CCID register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle). Note that, under normal conditions, the CCID register should only be updated from the command stream using the MI_SET_CONTEXT command.

DWord	Bit	Description
0	31:12	**Logical Render Context Address (LRCA)** Default Value: 0h Format: GraphicsAddress[31:12] This field contains the 4 KB-aligned Graphics Memory Address of the current Logical Rendering Context. Bit 11 MBZ.
	11:10	**Reserved** Format: MBZ

CCID - Current Context Register

9 — HD DVD Context

Value	Name	Description
0h	Regular Context	
1h	HD DVD Context	Special considerations for TDP allow for higher voltage and frequency.

8 — Reserved

Format:	Must Be One

7:4 — Reserved

Format:	MBZ

3 — Extended State Save Enable

Format:	Enable

If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, is saved as part of switching away from this logical context.

2 — Extended State Restore Enable

Format:	Enable

If set, the extended state identified in the Logical Context Data section of the Memory Data Formats chapter, was loaded (or restored) as part of switching to this logical context.

1 — Reserved

Format:	MBZ

0 — Valid

Format:	U1

Value	Name	Description
0h	Invalid **[Default]**	The other fields of this register are invalid. A switch away from the context will not invoke a context save operation.
1h	Valid	The other fields of this register are valid, and a switch from the context will invoke the normal context save/restore operations.

Current Idle/Busy/Avg Count for Freq Down Recommendation

RPCURDN - Current Idle/Busy/Avg Count for Freq Down Recommendation

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A060h-0A063h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Current Busy in Down EI** Access: RO Reports the current busyness in the down evaluation interval 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_current_ei_down_busy[23:0]

Current Idle/Busy/Avg Count for Freq Up Recommendation

RPCURUP - Current Idle/Busy/Avg Count for Freq Up Recommendation

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A054h-0A057h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Current Busy in UP EI** Access: RO Reports the current busyness in the UP evaluation interval 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_current_ei_up_busy[23:0]

Current Time in DOWN EI

RPCURDNEI - Current Time in DOWN EI		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A05Ch-0A05Fh	

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Current_Time_in_Down_EI** Access: RO Reports the current Time in the down evaluation interval 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_current_ei_down_time[23:0]

Current Time in UP EI

RPCURUPEI - Current Time in UP EI		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A050h-0A053h	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO
	23:0	**Current Time** Access: RO Reports the current time in UP EI 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_current_ei_up_time[23:0]

Customizable Event Creation 0-0

CEC0-0 - Customizable Event Creation 0-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02770h
Valid Projects:	CHV, BSW

This register is used to define custom counter event 0, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the last CEC block as the input bus to CEC0 block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC0-0 - Customizable Event Creation 0-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC0 comparator when comparing the compare value to the value active on the CEC0 conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 1-0

CEC1-0 - Customizable Event Creation 1-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02778h
Valid Projects:	CHV, BSW

This register is used to define custom counter event 1, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC1-0 - Customizable Event Creation 1-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 2-0

CEC2-0 - Customizable Event Creation 2-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02780h
Valid Projects:	CHV, BSW

This register is used to define custom counter event 2, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC2-0 - Customizable Event Creation 2-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 3-0

CEC3-0 - Customizable Event Creation 3-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02788h
Valid Projects:	CHV, BSW

This register is used to define custom counter event 3, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC3-0 - Customizable Event Creation 3-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 4-0

CEC4-0 - Customizable Event Creation 4-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02790h

This register is used to define custom counter event 4, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the last CEC block as the input bus to CEC0 block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC4-0 - Customizable Event Creation 4-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 5-0

CEC5-0 - Customizable Event Creation 5-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02798h

This register is used to define custom counter event 5, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Project** 0b / Pass-through / Input bit is passed through to comparator as is / All 1b / Negated / Input bit is negated before passing to comparator / All
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** / **Project** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block / All 11b / Reserved / /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC5-0 - Customizable Event Creation 5-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 6-0

CEC6-0 - Customizable Event Creation 6-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027A0h

This register is used to define custom counter event 6, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** / **Project** 0b / Pass-through / Input bit is passed through to comparator as is / All 1b / Negated / Input bit is negated before passing to comparator / All
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** / **Project** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block / All 11b / Reserved / /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC6-0 - Customizable Event Creation 6-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

Customizable Event Creation 7-0

CEC7-0 - Customizable Event Creation 7-0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	027A8h

This register is used to define custom counter event 7, bit definitions in this register refer to the CEC block diagram in the Custom Event Counters section.

DWord	Bit	Description
0	31:21	**Negate** Project: CHV, BSW Format: U11 The 11-bit array defined in this field allows input bus bits [10:0] to be individually negated in order to facilitate more complicated custom event creation (e.g. A & (!B | !C)). Note that LSB of this field affects bit 0 of the selected input bus. **Value** / **Name** / **Description** 0b / Pass-through / Input bit is passed through to comparator as is 1b / Negated / Input bit is negated before passing to comparator
	20:19	**Source Select** Format: U2 Selects the input signals to the Boolean event definition logic (see block diagram in the Custom Event Counters section). **Value** / **Name** / **Description** 01b / Prev Event / Selects the conditioned/flopped input from the previous CEC block as the input bus to this CEC block 11b / Reserved /
	18:3	**Compare Value** Format: U16 The value in this field is compared the 16-bit conditioned input bus that are fed into the comparator (see block diagram in the Custom Event Counters section). The type of comparison that is done is controlled by the Compare Function. When the compare function is true, then the signal for the custom event is asserted. This signal in turn can be counted by the B0 performance counter or fed into other CEC blocks.

CEC7-0 - Customizable Event Creation 7-0

	2:0	**Compare Function** Format: U3 This field selects the function used by the CEC comparator when comparing the compare value to the value active on the CEC conditioned input bus (see block diagram in the Custom Event Counters section).

Value	Name	Description
000b	Any Are Equal	Compare and assert output if any are equal (Can be used as OR function)
001b	Greater Than	Compare and assert output if greater than
010b	Equal	Compare and assert output if equal to (Can also be used as AND function)
011b	Greater Than or Equal	Compare and assert output if greater than or equal
100b	Less Than	Compare and assert output if less than
101b	Not Equal	Compare and assert output if not equal
110b	Less Than or Equal	Compare and assert output if less than or equal
111b	Reserved	

CVS TLB LRA 0

CVS_TLB_LRA_0 - CVS TLB LRA 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x5F201F00 CHV, BSW
Size (in bits):	32
Address:	04A20h

DWord	Bit	Description
0	31	**Reserved** Default Value: 0b Access: RO
	30:24	**CVS LRA1 Max** Default Value: 1011111b Project: CHV, BSW, :GT2:B Access: R/W Maximum value of programmable LRA1.
	23	**Reserved** Default Value: 0b Access: RO
	22:16	**CVS LRA1 Min** Default Value: 0100000b Access: R/W Minimum value of programmable LRA1.
	15	**Reserved** Default Value: 0b Access: RO
	14:8	**CVS LRA0 Max** Default Value: 0011111b Access: R/W Maximum value of programmable LRA0.
	7	**Reserved** Default Value: 0b Access: RO

CVS_TLB_LRA_0 - CVS TLB LRA 0		
	6:0	**CVS LRA0 Min** Default Value: 0000000b Access: R/W Minimum value of programmable LRA0.

CVS TLB LRA 1

CVS_TLB_LRA_1 - CVS TLB LRA 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW, :GT2:B
Source:	PRM
Default Value:	0x7F007F60
Size (in bits):	32
Address:	04A24h

DWord	Bit	Description
0	31	**Reserved** Default Value: 0b Access: RO
	30:24	**CVS LRA3 Max** Default Value: 1111111b Access: R/W For Future Use.
	23	**Reserved** Default Value: 0b Access: RO
	22:16	**CVS LRA3 Min** Default Value: 0000000b Access: R/W For Future Use.
	15	**Reserved** Default Value: 0b Access: RO
	14:8	**CVS LRA2 Max** Default Value: 1111111b Access: R/W Maximum value of programmable LRA2.
	7	**Reserved** Default Value: 0b Access: RO

CVS_TLB_LRA_1 - CVS TLB LRA 1

	6:0	**CVS LRA2 Min** Default Value: 1100000b Access: R/W Minimum value of programmable LRA2.

CVS TLB LRA 2

CVS_TLB_LRA_2 - CVS TLB LRA 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW, :GT2:B
Source:	PRM
Default Value:	0x0000001A
Size (in bits):	32
Address:	04A28h

DWord	Bit	Description
0	31:8	**Reserved** Default Value: 000000h Access: RO
	7:6	**RS LRA** Default Value: 00b Access: R/W Which LRA should RS use
	5:4	**CS LRA** Default Value: 01b Access: R/W Which LRA should CS use.
	3:2	**SOL LRA** Default Value: 10b Access: R/W Which LRA should SOL use.
	1:0	**VF LRA** Default Value: 10b Access: R/W Which LRA should VF use.

Depth/Early Depth TLB Partitioning Register

ZSHR - Depth/Early Depth TLB Partitioning Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000020
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04050h

This register is used to determine the number of TLB entries from the total of 64 available to be used by the Depth partition of the TLB. The rest of the entries are used for the Early Depth/Stencil TLB.

DWord	Bit	Description
0	31:6	**Reserved** Format: MBZ
	5:0	**Number of TLB Entries Out of 64 used for Depth TLB** Default Value: 32 The rest are be used for Early Depth/Stencil TLB. Default value is 32.

DID

DID - DID

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x22B08086 CHV, BSW
Size (in bits):	32
Address:	00000h

D2: PCI Device ID and Vendor ID Register

DWord	Bit	Description
0	31:24	**DEVICEID_UB** Default Value: 22h Access: R/W DID_MSB: Identifier assigned to the dev2 PCI. Punit will update this bitfield before handing control to BIOS.
	23:16	**DEVICEID_LB** Default Value: B0h Project: CHV, BSW Access: RO DID_LSB: Identifier assigned to the dev2 PCI. This bitfield is updated from metal straps. CHV, BSW allocated values : 0xB0 0xB1 0xB2 0xB3.
	23:16	**DEVICEID_LB** Default Value: B0h Project: CHV, BSW Access: RO DID_LSB: Identifier assigned to the dev2 PCI. Bits[7:2] is updated from metal straps. Bits[1:0] will be based on market segment SKU (ie via. fusing) CHV, BSW allocated values : 0xB0 0xB1 0xB2 0xB3.

DID - DID		
	15:0	**VENDORID** Default Value: 8086h Access: RO VID: PCI standard identification for Intel

DS Invocation Counter

DS_INVOCATION_COUNT - DS Invocation Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02308h

This register stores the number of domain points shaded by the DS threads. Domain points which hit in the DS cache will not cause this register to increment. Note that the spawning of a DS thread which shades two domain points will cause this counter to increment by two.This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**DS Invocation Count UDW** Number of domain points shaded by the DS threads. Updated only when DS Function Enable and Statistics Enable are set in 3DSTATE_DS
	31:0	**DS Invocation Count LDW** Number of domain points shaded by the DS threads. Updated only when DS Function Enable and Statistics Enable are set in 3DSTATE_DS

DX9 Constants Not Consumed By RCS

DX9CONST_PRODUCE_COUNT - DX9 Constants Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02484h

This register keeps track of the outstanding DX9 Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**DX9 Constants Produce Count** This register keeps track of the outstanding DX9 Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

DX9 Constants Prsed By RCS

DX9CONST_PARSE_COUNT - DX9 Constants Prsed By RCS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	02494h

This register keeps track of the DX9 Constant commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has DX9 Pool Alloc Valid. DX9 parse count should be less then equal to the DX9 produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. DX9 Parse count is subtracted from the DX9 Produce Count upon parsing 3D_PRIMITIVE command. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**DX9 Constants Produce Count** This register keeps track of the DX9 Constant commands parsed by RCS prior to encountering the Draw Call in an RS enabled Batch Buffer which has DX9 Pool Alloc Valid. DX9 parse count should be less then equal to the DX9 produce count for Command Streamer to make progress on a 3D_PRIMITIVE command. DX9 Parse count is subtracted from the DX9 Produce Count upon parsing 3D_PRIMITIVE command.

ECO Bits - Bus Reset Domain with lock bit

ECOBUS - ECO Bits - Bus Reset Domain with lock bit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A180h-0A183h

DWord	Bit	Description
0	31	**ECO Bits - Bus Reset Domain - LOCK BIT** Access: R/W Lock
	30	**Reserved** Access: RO
	29	**Flush and block gfx pipes during cpd enter** Access: R/W Lock 0 = GFX Pipes will not blocked during CPD enter 1 = GFX Pipes will be blocked and flushed during CPD enter. They will be unblocked again during CPD exit. This bit must not be set to 1
	28	**Block gfx from accessing memory during cpd enter** Access: R/W Lock 0 = GFX will not be blocked from accessing memory during CPD enter 1 = GFX will be blocked from accessing memory (go=0) during CPD enter. They will be unblocked again during CPD exit (go=1).
	27:26	**ECO Bits - Bus Reset Domain1** Access: R/W Lock Register bits that have no connection to design. Used to enable/disable changes to the design that were put in during the ECO process. pmcr_eco_busrst[26:25]
	25	**CPD GAM GO Messaging Enable** Access: R/W Lock 0 = No GO Messages to GAM during during CPD enter/exit flows (including slice shutdown) (default) 1 = GO Messages to GAM will occur during CPD enter/exit flows (including slice shutdown)

ECOBUS - ECO Bits - Bus Reset Domain with lock bit

	24:0	**ECO Bits - Bus Reset Domain0**

Access:	R/W Lock

Description

Register bits that have no connection to design. Used to enable/disable changes to the design that were put in during the ECO process.
pmcr_eco_busrst[24:0]
----
[21] = 1 VCS/CS/BCS Idleness, as well as CP/MBC/DT Idleness required for RC6 entry
[21] = 0 VCS/CS/BCS Idleness required for RC6 entry
----
[20] - Reserved
[19] - Reserved
[18] - Reserved
1 - Disable compensation
0 - Enable compensation of respective counters
----
[17] - Reserved
1 -- Disable WIDI Context Save/Restore
0 -- Enable WIDI Context Save/Restore (Default)

[16:15] -- Bits to indicate how many EU's, in each SubSlice, to bring up as part of Render Standby exit
00 -- Bring up 8 EU's (Default)
01 -- Bring up only 6 EU's
10 -- Bring up only 4 EU's
11 -- Bring up just 2 EU's

[14:0] are spare bits for ECO process.

ECO Bits - Device Reset Domain

ECODEV - ECO Bits - Device Reset Domain		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A184h-0A187h	
DWord	**Bit**	**Description**
0	31:0	**ECODEV** Access: R/W Register bits that have no connection to design. Used to enable/disable changes to the design that were put in during the ECO process. pmcr_eco_bits[31:0]

ECO Message Register

ECO_MSG - ECO Message Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08040h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15	**Placeholder for ECO Bit 15** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	14	**Placeholder for ECO Bit 14** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	13	**Placeholder for ECO Bit 13** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	12	**Placeholder for ECO Bit 12** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	11	**Placeholder for ECO Bit 11** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	10	**Placeholder for ECO Bit 10** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.

ECO_MSG - ECO Message Register

	9	**Placeholder for ECO Bit 9** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	8	**Placeholder for ECO Bit 8** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	7	**Placeholder for ECO Bit 7** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	6	**Placeholder for ECO Bit 6** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	5	**Placeholder for ECO Bit 5** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	4	**Placeholder for ECO Bit 4** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	3	**Placeholder for ECO Bit 3** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	2	**Placeholder for ECO Bit 2** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	1	**Placeholder for ECO Bit 1** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.
	0	**Placeholder for ECO Bit 0** Access: R/W ECO Bits 1'b0 : <default> Register definition will be modified if ECOs are required.

ECO Reserved

ECOResrv - ECO Reserved	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09898h

ECO Reserved bits

DWord	Bit	Description
0	31:0	**ECO Reserved Bits** Access: R/WC

ECREQ

ECREQ - ECREQ

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300A0h

This register contains the energy counter request bit.

DWord	Bit	Description
0	31:1	**SPARE** Default Value: 00000000h Access: R/W Spare bits for Duty cycle control feature.
	0	**Energy Count Request** Default Value: 0b Access: WO WO access type of this register means that it will always return a zero when read. A write will generate a one CZ clock wide pulse to indicate that Punit requests for Gunit to push the energy counters. This pulse will only be created when both the data attempted to be written is one and when the byte enables allow the write.

Element Descriptor Register

ELEM_DESCRIPTOR - Element Descriptor Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000
Access:	RO
Size (in bits):	64
Address:	04500h
Name:	BCS Element Descriptor Register
ShortName:	BCS_ELEM_DESCRIPTOR
Address:	04400h
Name:	RCS Element Descriptor Register
ShortName:	RCS_ELEM_DESCRIPTOR
Address:	04440h
Name:	VCS Element Descriptor Register
ShortName:	VCS_ELEM_DESCRIPTOR
Address:	044C0h
Name:	VECS Element Descriptor Register
ShortName:	VECS_ELEM_DESCRIPTOR

Element Information: The register is populated by command streamer and consumed by GAM

DWord	Bit	Description
0	63:32	**Context ID** Context identification number assigned to separate this context from others. Context IDs needs to be recycled in such a way that there could not be two active context with the same ID. This is a unique identification number by which a context is identified and referenced
	31:12	**LRCA** Command Streamer Only
	11:9	**Function Number** GFX device is considered to be on Bus0 with device number of 2. Function number is normally assigned as "0" however for gfx virtualization; there would be different function numbers which needs to be attached to context. Not used in Gen8.

ELEM_DESCRIPTOR - Element Descriptor Register

8	**Privileged Context / GGTT vs PPGTT mode** In Legacy Context: Defines the page tables to be used. This is how page walker come to know PPGTT vs GGTT selection for the entire context. In Advanced Context: Defines the privilege level for the context

Value	Name	Description
0h	**[Default]**	Use Global GTT (In Legacy Context) User Mode Context (In Advanced Context)
1h		Use Per-Process GTT (In Legacy Context) Supervisor Mode Context (In Advanced Context)

7:6	**Fault Model**

Value	Name	Description
00h	**[Default]**	Fault and Hang. Same mode as gen7.5
01h		Fault and Halt/Wait. Same mode as gen7.5
10h		Reserved
11h		Fault and Continue (reserved for gen8) - does not generate a page request to IOMMU.

5	**Deeper IA coherency Support** In Advanced Context: Defines the level of IA coherency

Value	Name	Description
0h	**[Default]**	IA coherency is provided at LLC level for all streams of GPU (i.e. gen7.5 like mode)
1h		IA coherency is provided at L3 level for EU data accesses of GPU

4	**A and D Support / 32 and 64b Address Support** In Legacy Context: Defines 32b vs 64b (48b canonical) addressing format In Advanced Context: Defines A/D bit support

Value	Name	Description
0h	**[Default]**	32b addressing format (In Legacy Context) A/D bit management in page tables is NOT supported (In Advanced Context)
1h		64b (48b canonical) addressing format (In Legacy Context) A/D bit management in page tables is supported (In Advanced Context)

3	**Context Type: Legacy vs Advanced** Defines the context type. Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected.

Value	Name	Description
0h	**[Default]**	Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models...). Note that advanced context is not bounded to GPGPU.
1h		Legacy Context: Defines the context as legacy mode which is similar to prior generations of gen8.

ELEM_DESCRIPTOR - Element Descriptor Register

	2	**FR** Command Streamer Specific
	1	**Scheduling Mode** Project:

Value	Name	Description
0h	**[Default]**	Indicates execlist mode of scheduling.
1h		Indicates Ring Buffer mode of scheduling.

	0	**Valid** Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it will continue but flag an error.

Error_Identity_Reg

EIR - Error_Identity_Reg	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	020B0h-020B3h
Name:	Error Identity Register
ShortName:	EIR_RCSUNIT
Address:	120B0h-120B3h
Name:	Error Identity Register
ShortName:	EIR_VCSUNIT0
Address:	1A0B0h-1A0B3h
Name:	Error Identity Register
ShortName:	EIR_VECSUNIT
Address:	1C0B0h-1C0B3h
Name:	Error Identity Register
ShortName:	EIR_VCSUNIT1
Address:	220B0h-220B3h
Name:	Error Identity Register
ShortName:	EIR_BCSUNIT
Address:	1820B0h

Error Identity. This register contains the persistent values of ESR error status bits that are unmasked via the EMR register.
The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register (ISR).
In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field.
If required, software should then proceed to clear the Master Error bit of the IIR

DWord	Bit	Description
0	31:7	**RESERVED** Default Value: 0000000h Access: RO Reserved
	6	**Gunit_TLB_DataE** Default Value: 0b

EIR - Error_Identity_Reg

	Access:	R/W One Clear
	Set if Gunit TLB has a data valid error	
5	**Gunit_TLB_PTE**	
	Default Value:	0b
	Access:	R/W One Clear
	Set if Gunit TLB has a PTE translation error.	
4	**PAGE_TABLE_ERROR**	
	Default Value:	0b
	Access:	R/W One Clear
	PTE: This bit is set when a Graphics Memory Mapping Error is detected and it's not EMR masked. The cause of the error is indicated (to some extent) in the PGTBL_ER register. This error condition cannot be cleared except by reset (i.e., it is a fatalerror) mir_EIR[4] = (((dsp_gvd_invldgttpte_int_dczfwoh and ~mir_EMR[4]) | mir_EIR[4]) and ~(sys_wdata[4] and mir_write and mirb0_decode and ~sys_data_mask[0]));	
3:1	**RESERVED**	
	Default Value:	0h
	Access:	RO
	Reserved	
0	**CLAIM_ERROR**	
	Default Value:	0b
	Access:	R/W One Clear
	If EMR[0]=1, this bit is set when an address within Gunit address space is accessed, but no memory mapped register exists in this address. This error condition can be cleared by writing 1b to this bit. Mir_EIR[0] = (((noRMclaim_err and ~mir_EMR[0]) |mir_EIR[0]) and ~(sys_wdata[0] and mir_write and mirb0_decode and ~sys_data_mask[0]));	

Error_Mask_Reg

EMR - Error_Mask_Reg

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000071
Size (in bits):	32

Address:	020B4h-020B7h
Name:	Error Mask Register
ShortName:	EMR_RCSUNIT

Address:	120B4h-120B7h
Name:	Error Mask Register
ShortName:	EMR_VCSUNIT0

Address:	1A0B4h-1A0B7h
Name:	Error Mask Register
ShortName:	EMR_VECSUNIT

Address:	1C0B4h-1C0B7h
Name:	Error Mask Register
ShortName:	EMR_VCSUNIT1

Address:	220B4h-220B7h
Name:	Error Mask Register
ShortName:	EMR_BCSUNIT

Address:	1820B4h

Error Mask. This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR

DWord	Bit	Description
0	31:7	**RESERVED** Default Value: 0000000h Access: RO Reserved
	6	**Gunit_TLB_DataE** Default Value: 1b Access: R/W 1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.

EMR - Error_Mask_Reg

	5	**Gunit_TLB_PTE** Default Value: 1b Access: R/W 1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.
	4	**PAGE_TABLE_ERROR** Default Value: 1b Access: R/W 1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.
	3:1	**RESERVED** Default Value: 0h Access: RO Reserved
	0	**CLAIM_ERROR** Default Value: 1b Access: R/W 1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.

Error_Status_Reg

ESR - Error_Status_Reg	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	020B8h-020BBh
Name:	Error Status Register
ShortName:	ESR_RCSUNIT
Address:	120B8h-120BBh
Name:	Error Status Register
ShortName:	ESR_VCSUNIT0
Address:	1A0B8h-1A0BBh
Name:	Error Status Register
ShortName:	ESR_VECSUNIT
Address:	1C0B8h-1C0BBh
Name:	Error Status Register
ShortName:	ESR_VCSUNIT1
Address:	220B8h-220BBh
Name:	Error Status Register
ShortName:	ESR_BCSUNIT
Address:	1820B8h

Error Status. This register contains the non-persistent values of all hardware-detected error condition bits.

DWord	Bit	Description
0	31:7	**RESERVED** Default Value: 0000000h Access: RO Reserved
	6	**Gunit_TLB_DataE** Default Value: 0b Access: RO 1 = This error condition currently exists. 0 = This error condition currently does not exist

ESR - Error_Status_Reg

	5	**Gunit_TLB_PTE** Default Value: 0b Access: RO 1 = This error condition currently exists. 0 = This error condition currently does not exist
	4	**PAGE_TABLE_ERROR** Default Value: 0b Access: RO 1 = This error condition currently exists. 0 = This error condition currently does not exist(Was reported based on 0x18_2024h "OR".
	3:1	**RESERVED** Default Value: 0h Access: RO Reserved
	0	**CLAIM_ERROR** Default Value: 0b Access: RO No RMbus claim occurred. Logging the error. 1 = This error condition currently exists. 0 = This error condition currently does not exist. Internal Gunit 'noRMclaim_err' wire from RMbus master.

Error Identity Register

EIR - Error Identity Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W, RO
Size (in bits):	32
Address:	020B0h

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a 1 to the appropriate bit(s)), except for the unrecoverable bits described.)

Restriction

Restriction EIR register contents are not power or render context save/restored. EIR register contents of an engine will get lost when the corresponding graphics engine (Render, Video, Vidoe Enhancement, Blitter) is power down.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Error Identity Bits** Format: Array of Error condition bits See the table titled Hardware-Detected Error Bits. This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See Table Table 3-3. Hardware-Detected Error Bits). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR. Reserved bits are RO. **Value** / **Name** 1h — Error occurred **Programming Notes** Writing a 1 to a set bit will cause that error condition to be cleared. However, neither the Page Table Error bit (Bit 4) nor the Instruction Error bit (Bit 0) can be cleared except by reset (i.e., it is a fatal error).

Error Mask Register

EMR - Error Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0xFFFFFFFF
Access:	R/W, RO
Size (in bits):	32
Address:	020B4h

The EMR register is used by software to control which Error Status Register bits are masked or unmasked. Unmasked bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. Masked bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts. Reserved bits are RO.

DWord	Bit	Description
0	31:8	**Reserved** Default Value: FFFFFFh Format: Must Be One **Programming Notes** These bits are not implemented in HW and must be set to '1'
	7:0	**Error Mask Bits** Format: Array of error condition mask bits See the table titled Hardware-Detected Error Bits. This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Value	Name	Description
FFh	**[Default]**	
0h	Not Masked	Will be reported in the EIR
1h	Masked	Will not be reported in the EIR

Error Reporting Register

ERR - Error Reporting Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B42Ch	

DWord	Bit	Description
0	31:5	**Reserved** Access: RO Reserved.
	4	**First Content Buffer Ready 0** Access: R/W First Content Buffer Ready 0 (FRSNTBFR0). First Content Buffer Ready: This bit gets set by the HW when the buffer is completely filled up and cleared by the driver when the contents of this buffer are copied out of memory. Is set by lpfc_lpconf_buffer0_ready (pulse). lpconf_lpfc_buffer0_ready (static signal to lpfc).
	3	**Second Buffer ready slice 0** Access: R/W Second Content Buffer Ready slice 0 (SCNBFR0). Second Content Buffer Ready: This bit gets set by the HW when the buffer is completely filled up and cleared by the driver when the contents of this buffer are copied out of memory. Is set by lpfc_lpconf_buffer1_ready (pulse). lpconf_lpfc_buffer1_ready (static signal to lpfc).
	2	**Write Expire Error Slice 0** Access: R/W Write Expired Error slice 0 (WEERR0). Write Expired Error: If DMA controller could not get a chance to push the write of 64Bytes to LTISEQ and data gets clobbered with the new expiration of the save timer, this error bit is set to indicate something went wrong. Signal -lpfc_lpconf_wrexp_error.
	1	**Buffer full Error Slice 0** Access: R/W Buffer full Error Slice 0 (BFFLERR0). Set by lpfc_lpconf_error_buffer_full. When all buffers are full lpfc sets this bit or if only 1 buffer is enabled then lpfc sets this bit when the buffer is full.

ERR - Error Reporting Register			
	0	**Reserved**	
		Access:	RO
		Reserved.	

Error Status Register

ESR - Error Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	020B8h

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition persistent). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Error Status Bits** Format: Array of error condition bits See the table titled Hardware-Detected Error Bits. This register contains the non-persistent values of all hardware-detected error condition bits. Value: 1h — Name: Error Condition Detected

EU Mask Programming

TD_PM_MODE_EUCOUNT - EU Mask Programming

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32

Address:	0E4F8h
Name:	EU Mask Programming Slice 0
ShortName:	TD_PM_MODE_EUCOUNT_S0
Valid Projects:	CHV, BSW

Address:	0E5F8h
Name:	EU Mask Programming Slice 1
ShortName:	TD_PM_MODE_EUCOUNT_S1
Valid Projects:	CHV, BSW

Address:	0E6F8h
Name:	EU Mask Programming Slice 2
ShortName:	TD_PM_MODE_EUCOUNT_S2
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:24	**Reserved** Project: CHV, BSW Format: MBZ
	23	**SubSlice 2 EU 7 Enable** Format: Enable
	22	**SubSlice 2 EU 6 Enable** Format: Enable
	21	**SubSlice 2 EU 5 Enable** Format: Enable
	20	**SubSlice 2 EU 4 Enable** Format: Enable
	19	**SubSlice 2 EU 3 Enable** Format: Enable
	18	**SubSlice 2 EU 2 Enable**

TD_PM_MODE_EUCOUNT - EU Mask Programming			
		Format:	Enable
	17	**SubSlice 2 EU 1 Enable**	
		Format:	Enable
	16	**SubSlice 2 EU 0 Enable**	
		Format:	Enable
	15	**SubSlice 1 EU 7 Enable**	
		Format:	Enable
	14	**SubSlice 1 EU 6 Enable**	
		Format:	Enable
	13	**SubSlice 1 EU 5 Enable**	
		Format:	Enable
	12	**SubSlice 1 EU 4 Enable**	
		Format:	Enable
	11	**SubSlice 1 EU 3 Enable**	
		Format:	Enable
	10	**SubSlice 1 EU 2 Enable**	
		Format:	Enable
	9	**SubSlice 1 EU 1 Enable**	
		Format:	Enable
	8	**SubSlice 1 EU 0 Enable**	
		Format:	Enable
	7	**SubSlice 0 EU 7 Enable**	
		Format:	Enable
	6	**SubSlice 0 EU 6 Enable**	
		Format:	Enable
	5	**SubSlice 0 EU 5 Enable**	
		Format:	Enable
	4	**SubSlice 0 EU 4 Enable**	
		Format:	Enable
	3	**SubSlice 0 EU 3 Enable**	
		Format:	Enable
	2	**SubSlice 0 EU 2 Enable**	
		Format:	Enable
	1	**SubSlice 0 EU 1 Enable**	
		Format:	Enable

TD_PM_MODE_EUCOUNT - EU Mask Programming			
	0	**SubSlice 0 EU 0 Enable**	
		Format:	Enable

EU Metrics for Event0 High

EUMETRICSEVENT0H - EU Metrics for Event0 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138134h	
Upper 32 bits of the EU Metrics Event0		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event0 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event0

EU Metrics for Event0 Low

EUMETRICSEVENT0L - EU Metrics for Event0 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138130h

Lower 14 bits of the EU Metrics Event0

DWord	Bit	Description
0	31:18	**EUMetrics, Event0 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event0
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event1 High

EUMETRICSEVENT1H - EU Metrics for Event1 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	13813Ch	
Upper 32 bits of the EU Metrics Event1		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event1 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event1

EU Metrics for Event1 Low

EUMETRICSEVENT1L - EU Metrics for Event1 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138138h

Lower 14 bits of the EU Metrics Event1

DWord	Bit	Description
0	31:18	**EUMetrics, Event1 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event1
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event2 High

EUMETRICSEVENT2H - EU Metrics for Event2 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138144h	
Upper 32 bits of the EU Metrics Event2		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event2 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event2

EU Metrics for Event2 Low

EUMETRICSEVENT2L - EU Metrics for Event2 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138140h

Lower 14 bits of the EU Metrics Event2

DWord	Bit	Description
0	31:18	**EUMetrics, Event2 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event2
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event3 High

EUMETRICSEVENT3H - EU Metrics for Event3 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	13814Ch	
Upper 32 bits of the EU Metrics Event3		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event3 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event3

EU Metrics for Event3 Low

EUMETRICSEVENT3L - EU Metrics for Event3 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138148h

Lower 14 bits of the EU Metrics Event3

DWord	Bit	Description
0	31:18	**EUMetrics, Event3 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event3
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event4 High

EUMETRICSEVENT4H - EU Metrics for Event4 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138154h	
Upper 32 bits of the EU Metrics Event4		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event4 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event4

EU Metrics for Event4 Low

EUMETRICSEVENT4L - EU Metrics for Event4 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138150h

Lower 14 bits of the EU Metrics Event4

DWord	Bit	Description
0	31:18	**EUMetrics, Event4 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event4
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event5 High

EUMETRICSEVENT5H - EU Metrics for Event5 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	13815Ch	
Upper 32 bits of the EU Metrics Event5		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event5 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event5

EU Metrics for Event5 Low

EUMETRICSEVENT5L - EU Metrics for Event5 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138158h

Lower 14 bits of the EU Metrics Event5

DWord	Bit	Description
0	31:18	**EUMetrics, Event5 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event5
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event6 High

EUMETRICSEVENT6H - EU Metrics for Event6 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138164h	
Upper 32 bits of the EU Metrics Event6		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event6 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event6

EU Metrics for Event6 Low

EUMETRICSEVENT6L - EU Metrics for Event6 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138160h

Lower 14 bits of the EU Metrics Event6

DWord	Bit	Description
0	31:18	**EUMetrics, Event6 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event6
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event7 High

EUMETRICSEVENT7H - EU Metrics for Event7 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	13816Ch	
Upper 32 bits of the EU Metrics Event7		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event7 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event7

EU Metrics for Event7 Low

EUMETRICSEVENT7L - EU Metrics for Event7 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138168h

Lower 14 bits of the EU Metrics Event7

DWord	Bit	Description
0	31:18	**EUMetrics, Event7 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event7
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event8 High

EUMETRICSEVENT8H - EU Metrics for Event8 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138174h	
Upper 32 bits of the EU Metrics Event8		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event8 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event8

EU Metrics for Event8 Low

EUMETRICSEVENT8L - EU Metrics for Event8 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138170h

Lower 14 bits of the EU Metrics Event8

DWord	Bit	Description
0	31:18	**EUMetrics, Event8 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event8
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event9 High

EUMETRICSEVENT9H - EU Metrics for Event9 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	13817Ch	
Upper 32 bits of the EU Metrics Event9		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event9 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event9

EU Metrics for Event9 Low

EUMETRICSEVENT9L - EU Metrics for Event9 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138178h

Lower 14 bits of the EU Metrics Event9

DWord	Bit	Description
0	31:18	**EUMetrics, Event9 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event9
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event10 High

EUMETRICSEVENT10H - EU Metrics for Event10 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138184h	
Upper 32 bits of the EU Metrics Event10		
DWord	**Bit**	**Description**
0	31:0	**EUMetrics, Event10 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event10

EU Metrics for Event10 Low

EUMETRICSEVENT10L - EU Metrics for Event10 Low		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	138180h	
Lower 14 bits of the EU Metrics Event10		
DWord	**Bit**	**Description**
0	31:18	**EUMetrics, Event10 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event10
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event11 High

EUMETRICSEVENT11H - EU Metrics for Event11 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	13818Ch

Upper 32 bits of the EU Metrics Event11

DWord	Bit	Description
0	31:0	**EUMetrics, Event11 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event11

EU Metrics for Event11 Low

EUMETRICSEVENT11L - EU Metrics for Event11 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138188h

Lower 14 bits of the EU Metrics Event11

DWord	Bit	Description
0	31:18	**EUMetrics, Event11 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event11
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

EU Metrics for Event12 High

EUMETRICSEVENT12H - EU Metrics for Event12 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138194h

Upper 32 bits of the EU Metrics Event12

DWord	Bit	Description
0	31:0	**EUMetrics, Event12 High** Default Value: 00000000h Access: RO Contains Upper 32 bits of EU Metrics, Event12

EU Metrics for Event12 Low

EUMETRICSEVENT12L - EU Metrics for Event12 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138190h

Lower 14 bits of the EU Metrics Event12

DWord	Bit	Description
0	31:18	**EUMetrics, Event12 Low** Default Value: 0000h Access: RO Contains lower 14 bits of EU Metrics, Event12
	17:0	**Reserved** Default Value: 00000h Access: RO Reserved

Event selection and base counters

LPFCREG2 - Event selection and base counters

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B00Ch

DWord	Bit	Description
0	31:24	**Counter 7 client** Access: R/W Incf_lpfc_cnt7_client[7:0].Client Encoding (hex): GAFS Rd 00 GAFS Wr 01 HDC0 Data Rd 02 HDC0 Const Rd 03 HDC0 URB Rd 04 HDC0 Data Wr 05 HDC0 URB Wr 06 HDC1 Data Rd 07 HDC1 Const Rd 08 HDC1 URB Rd 09 HDC1 Data Wr 0A HDC1 URB Wr 0B TDL0 Rd 0C TDL1 Rd 0D Tex0 Rd 0E Tex1 Rd 0F Tex2 Rd (reserved) 10 Tex3 Rd (reserved) 11 SBE Rd 12 IC0 Rd 13 IC1 Rd 14 SARB Rd 15 Aggregated Tex 16 SLM0 Rd 17 SLM1 Rd 18 SLM0 Wr 19 SLM1 Wr 1A SLM0 Atomics 1B SLM1 Atomics 1C Reserved 1D Reserved 1E

LPFCREG2 - Event selection and base counters

		Reserved 1F FF Stalls 20 HDC Stalls 21 TDL Stalls 22 Texture Stalls 23 IC Stalls 24 SBE Stalls 25 SLM Stalls 26 Bank0 Total Hits 40 Bank0 Total Cycles 41 Bank0 Total Rds 42 Bank0 Total Wrs 43 Bank0 FF Rds 44 Bank0 FF Wrs 45 Bank0 DC Rds 46 Bank0 DC Wrs 47 Bank0 DC Hits 48 rsvd 49 Bank0 Tex Rds 4A Bank0 Tex Hits 4B Bank0 IC Rds 4C Bank0 IC Hits 4D Reserved 4E Reserved 4F Bank1 Events 50-5F (except 59-reserved) Bank2 Events 60-6F(except 69-reserved) Bank3 Events 70-7F(except 79-reserved) MSC Rd 80 MSC Wr 81 STC Rd 82 STC Wr 83 Hiz Rd 84 Hiz Wr 85 RCZ Rd 86 RCZ Wr 87 RCC Rd 88 RCC Wr 89 LTCD0 Err Corr EE LTCD1 Err Corr EF LTCD2 Err Corr F0 LTCD3 Err Corr F1 LTCD0 Err UnCorr F2 LTCD1 Err UnCorr F3 LTCD2 Err UnCorr F4 LTCD3 Err UnCorr F5

LPFCREG2 - Event selection and base counters

		Counter#7 Client Selection: This field controls which client's request stream is observed in counter#7.
	23:16	**Counter 6 client** Access: R/W lncf_lpfc_cnt6_client[7:0].Counter#6 Client Selection: This field controls which client's request stream is observed in counter#6.
	15:8	**Counter 5 client** Access: R/W lncf_lpfc_cnt5_client[7:0].Counter#5 Client Selection: This field controls which client's request stream is observed in counter#5.
	7:0	**Counter 4 client** Access: R/W lncf_lpfc_cnt4_client[7:0].Counter#4 Client Selection: This field controls which client's request stream is observed in counter#4.

Event Selection and Base Counters1

LPFCREG1 - Event Selection and Base Counters1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B010h

DWord	Bit	Description
0	31:24	**Counter 3 client** Access: R/W Incf_lpfc_cnt3_client[7:0].Counter#3 Client Selection: This field controls which client's request stream is observed in counter#3.
	23:16	**Counter 2 client** Access: R/W Incf_lpfc_cnt2_client[7:0].Counter#2 Client Selection: This field controls which client's request stream is observed in counter#2.
	15:8	**Counter 1 Client** Access: R/W Incf_lpfc_cnt1_client[7:0].Counter#1 Client Selection: This field controls which client's request stream is observed in counter#1.
	7:0	**Counter0 Client** Access: R/W Incf_lpfc_cnt0_client[7:0].Counter#0 Client Selection: This field controls which client's request stream is observed in counter#0.

Execlist Status

EXECLIST_STATUS - Execlist Status	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001, 0x00000000
Access:	RO
Size (in bits):	64
Address:	02234h-0223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_RCSUNIT
Address:	12234h-1223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VCSUNIT0
Address:	1A234h-1A23Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VECSUNIT
Address:	1C234h-1C23Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_VCSUNIT1
Address:	22234h-2223Bh
Name:	RCS Execlist Status
ShortName:	EXECLIST_STATUS_BCSUNIT

This register contains the pointers and full indicator for the Execlist Queue and the context ID of the currently running context. DefaultValue = UUUU UUU1h (4:0 default to 00001b, others UNDEFINED).

DWord	Bit	Description
0	63:32	**Current Context ID** Format: U32 Contains the context ID of the currently running context.
	31:30	**Reserved** Project: CHV, BSW Format: MBZ
	29:27	**Reserved** Format: MBZ

EXECLIST_STATUS - Execlist Status

26:19	**Reserved** Project: CHV, BSW Format: MBZ
18	**Reserved**
17	**Reserved**
16	**Arbitration Enable** Project: CHV, BSW Format: U1 This field reflects the Arbitration Flag set by the MI_ARB_ON_OFF command in Command Streamer.
15:14	**Current Active Element Status** Project: CHV, BSW Format: U2 Points at the element being executed in current Execlist (if there is one). 00b: No Active Element being executed 01b: Element0 of current execlist being executed 10b: Element1 of current execlist being executed 11b: Reserved
13:5	**Last Context Switch Reason** Project: CHV, BSW Access: R/W Format: U9 This field contains the switch reason for the last context to switch away, as captured in the Context Status Dword, bits 8:0. **Programming Notes** This field should not written by SW.
4	**Execlist 0 Valid** Project: CHV, BSW Format: Flag This bit is set when the first DW for this Execlist port 0 is written through the submission port, and will not be cleared till the CSB is updated and the command stream is switching to the next execution list. If no execution list is pending, the transition of this bit from one to zero guarentees there will be no preemption on the next submission. 0: Invalid **[Default]** 1: Valid
3	**Execlist 1 Valid**

EXECLIST_STATUS - Execlist Status

		Project: CHV, BSW Format: Flag This bit is set when the first DW for this Execlist port 1 is written through the submission port, and will not be cleared till the CSB is updated and the command stream is switching to the next execution list. If no execution list is pending, the transition of this bit from one to zero guarentees there will be no preemption on the next submission. **Value** / **Name** 0 – Invalid **[Default]** 1 – Valid
	2	**Execlist Queue Full** Project: CHV, BSW When [Execlist Write Pointer] and [Current Execlist Pointer] are equal, this bit differentiates between Queue Full and Queue Empty. **Value** / **Name** / **Description** 0 – Execlist Queue Empty **[Default]** 1 – Execlist Queue Full – There is a current and a pending execlist.
	1	**Execlist Write Pointer** Project: CHV, BSW Format: ExeclistContentsIndex Determines which Execlist will be the next submitted to. When a new execlist is submitted, this pointer increments to point to the next execlist slot.
	0	**Current Execlist Pointer** Default Value: 1h Project: CHV, BSW Format: ExeclistContentsIndex Points at the currently executing Execlist (if there is one). This pointer advances when the first context of new execlist is restored.

Execlist Submit Port Register

EXECLIST_SUBMITPORT - Execlist Submit Port Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	02230h-02233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_RCSUNIT
Address:	12230h-12233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VCSUNIT0
Address:	1A230h-1A233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VECSUNIT
Address:	1C230h-1C233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_VCSUNIT1
Address:	22230h-22233h
Name:	Execlist Submit Port Register
ShortName:	EXECLIST_SUBMITPORT_BCSUNIT

SW should submit a new pending execlist to this register. The DWs of the context descriptors must be written in a specific order: Element 1 must be written first and then Element 0. For each Element, DW1 must be written first followed by DW0. Context descriptors for both the elements must be written even if only one context are being submitted. The valid bits of the unused context descriptors should be set to 0.

Order of DW Submission to the Execlist Port
Element 1, High Dword
Element 1, Low Dword
Element 0, High Dword
Element 0, Low Dword

If a execlist of only one element is being submitted, it must be submitted in Element 0. It is UNDEFINED to submit a execlist with the valid bit of Element 0 clear (an "empty" execlist). It is possible that one or all of the contexts submitted in a execlists are "empty"; that is, have head and tail pointers equal to each other indicating no commands to be run. All of the valid bits in the Execlist Element Status Registers for the "about to be submitted" execlist will be cleared when the first DW (DW1 of Element 1) is written to the submit port.

EXECLIST_SUBMITPORT - Execlist Submit Port Register

Submission of the Element 0 Context Descriptor low Dword with the valid bit set is interpreted as a request to switch (as soon as possible) to the new execlist, i.e., a pre-emption request.
If a submitted Execlist's Element 0 Context Descriptor LRCA matches the LRCA of the currently executing context, then the newly submitted execlist will become the currently executing execlist without any context switch and without any impact to the executing context except that it will re-sample the tail pointer from the context image. This is done in case more commands have been inserted into its ring buffer between the first execlist submission and the 2nd.

Programming Notes	Source
SW must ensure the contexts submitted to the both the context descriptors in the execlist are different, i.e SW must not submit the same context descriptor to both the elements of the execlist.	
SW must follow below programming sequence for ELSP submission in host mode of. SW must set Force Wakeup bit to prevent GT from entering C6 while ELSP writes are in progress. Ex: Set Force Wakeup Program ELSP writes Reset Force Wakeup	
Render CS Only: Command Streamer triggers IDLE sequence flows for RDOP_CG on un-successful Semaphore Waits and Wait for Display Events when "Inhibit Synchronous Context Switch" is set in CTXT_SR_CTL register. As part of IDLE flows CS flushes the Write Caches (Z, Color, HDC). While IDLE flush in progress context switch can happen due to pending execlist submitted and when this condition occurs CS might not issue context switch flush resulting in RO caches not invalidated (State, Texture, Instruction, Constant). To WA above issue, SW must always program a valid BB_PER_CTX_PTR for every context submitted with a PIPE_CONTROL command to invalidate all RO caches. PIPE_CONTROL with below bits set in it: Instruction Cache Invalidate Enable Texture Cache Invalidate Enable Constant Cache Invalidate Enable State Cache Invalidate Enable	RenderCS
[All Command Streamers]: When SW intends to use semaphore signaling between Command streamers, SW must avoid lite restores in HW by programming Force Restore bit to '1' in context descriptor during context submission, this is required to avoid known HW issue.	

Workaround	Source
Workaround: SW must always ensure there are valid commands to be executed by HW on a context submission, i.e ring buffer head pointer must not be equal to the ring buffer head pointer on context submission to HW for execution.	RenderCS

EXECLIST_SUBMITPORT - Execlist Submit Port Register

Additional Note: This WA need not be applied when the arbitration is not disabled prior to executing “Batch Buffer Per Context Pointer” as part of context restore. Arbitration can be disabled prior to executing “Batch Buffer Per Context Pointer” by programming MI_ARB_ON_OFF (arbitration disable) in indirect context pointer.	
Workaround: SW must always ensure a preempted context submitted to HW doesn’t undergo lite restore due to the same context getting submitted on the next Execlist submission. This can be achieved by setting “Force Restore Bit” in the context descriptor of the context getting submitted and if the same context is known to be submitted to HW for execution on the earlier Execlist submission Or SW on submitting a **preempted** context must wait for the context to switch out before submitting the same context to the Execlist Submit Port. **Note:** This WA need not be applied when “Force Sync Command Ordering” bit of INSTPM register is not disabled (programmed to value ‘0’) during execution of “Batch Buffer Per Context Pointer” during context restore. “Force Sync Command Ordering” can be disabled prior to or during execution of “Batch Buffer Per Context Pointer” by programming INSTPM register using MI_LOAD_REGISTER_IMM command in Indirect Context Pointer or in “Batch Buffer Per Context Pointer”. Disabling of “Force Sync Command Ordering” during “Batch Buffer Per Context Pointer” execution was required to address Resource Streamer related preemption issue on HSD 1912487, this WA is not applied when Resource Streamer (RS) is not enabled or when a Resource Streamer enabled context is not preemptable.	

DWord	Bit	Description
0	31:0	**Context Descriptor DW** Format: Context Descriptor See "Context Descriptor Format" for format. The element that this DW is submitted as and whether it is the high DW or the low DW is determined by order. This register must be written to 4 times in order to submit a execlist.

Execute Condition Code Register

EXCC - Execute Condition Code Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1
Address:	02028h
Address:	12028h-1202Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VCSUNIT0
Address:	1A028h-1A02Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VECSUNIT
Address:	1C028h-1C02Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_VCSUNIT1
Address:	22028h-2202Bh
Name:	Execute Condition Code Register
ShortName:	EXCC_BCSUNIT

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a 1, while instruction is discarded if the condition evaluates to a 0. Once excluded a ring is enabled into arbitration when the selected condition evaluates to a 0.
This register also contains control for the invalidation of indirect state pointers on context restore.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] These bits serves as a write enable for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15	**Reserved** Project: CHV, BSW Format: MBZ

EXCC - Execute Condition Code Register

	14	**Context Wait for V-blank on Pipe-C** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe C Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	13	**Context Wait for V-blank on Pipe-B** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe B Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	12	**Context Wait for V-blank on Pipe-A** Project: CHV, BSW This field when set indicates the corresponding context has executed MI_WAIT_FOR_EVENT with "Display Pipe A Vertical Blank Wait Enable" set. This is an internal HW flag and should not be accessed by SW.
	11	**Pending Indirect State Dirty Bit** Project: CHV, BSW Access: RO This field keeps track of whether or not an indirect state pointer command has been parsed in the current context. Clears either on a context save or explicitly through a flush command. This bit is Read Only.
	10:7	**Pending Indirect State Counter** Project: CHV, BSW This field keeps track of the maximum number of indirect state pointers pending in the system. When the register is saved/restored, it saves either a value of 1 or 0. This field is Read-Only.
	6:5	**Reserved** Format: MBZ
	4:0	**User Defined Condition Codes** The software may signal a Stream Semaphore by setting the Mask bit and Signal Bit together to match the bit field specified in a WAIT_FOR_EVENT (Semaphore).

FAULT_TLB_RD_DATA0 Register

FAULT_TLB_RD_DATA0 - FAULT_TLB_RD_DATA0 Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04B10h	
DWord	**Bit**	**Description**
0	31:0	**FAULT_TLB_READ_DATA0 Register** Default Value: 00000000h Access: RO Fault cycle Virtual address [43:12]

FAULT_TLB_RD_DATA1 Register

FAULT_TLB_RD_DATA1 - FAULT_TLB_RD_DATA1 Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B14h

DWord	Bit	Description
0	31:0	**FAULT_TLB_READ_DATA1 Register** Default Value: 00000000h Access: RO Bit[31:5] Reserved Bit[4] Cycle GTT SEL (1-GGTT Cycle, 0-PPGTT Cycle) Bit[3:0] Fault cycle Virtual address [47:44]

Fault Switch Out

FAULT_SO - Fault Switch Out			
Register Space:	MMIO: 0/2/0		
Project:	CHV		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	04590h		
DWord	**Bit**	**Description**	
0	31:0	**Fault Switch Out**	
		Default Value:	00000000h
		Access:	R/W

FBC_RT_BASE_ADDR_REGISTER

FBC_RT_BASE_ADDR_REGISTER - FBC_RT_BASE_ADDR_REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07020h
Valid Projects:	CHV, BSW

This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:12	**FBC RT Base Address** Access: R/W Format: PPGraphicsAddress[31:12] 4KB aligned Base Address as mapped in the PPGTT or in the GGTT for the render target. This base address must be the one that is either front buffer or the back-buffer (a flip target). It must be programmed before any draw call binding that render target base address.
	11:2	**Reserved** Access: R/W Format: PBC
	1	**FBC Front Buffer Target** Project: CHV, BSW Access: R/W Format: Enable

Value	Name	Description
0h	**[Default]**	FBC is targeting the Back Buffer for compression. This buffer can be cached in the MLC/LLC, so a GFDT flush is required before FBC can begin compression.
1h		FBC is targeting the Font Buffer for compression. This buffer cannot be cached in the MLC/LLC. FBC compression can begin after any RC flush.

FBC_RT_BASE_ADDR_REGISTER - FBC_RT_BASE_ADDR_REGISTER

	0	**PPGTT Render Target Base Address Valid for FBC**

Project:	CHV, BSW
Access:	R/W
Format:	Enable

Value	Name	Description
0h	**[Default]**	Base address in this register [31:12] is not valid and therefore FBC will not get any modifications from rendering.
1h		Base address in this register [31:12] is valid and HW needs to compare the current render target base address with this base address to provide modifications to FBC.

FBC_RT_BASE_ADDR_REGISTER_UPPER

FBC_RT_BASE_ADDR_REGISTER_UPPER - FBC_RT_BASE_ADDR_REGISTER_UPPER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07024h

This Register is saved and restored as part of Context.

DWord	Bit	Description
0	31:16	**Reserved** Access: R/W Format: PBC
	15:0	**FBC RT Base Address High** Access: R/W Format: BaseAddress[47:32] Must be set to modify corresponding data bit. Reads to this field returns zero. Upper 4KB aligned Base Address as mapped in the PPGTT or in the GGTT for the render target. This base address must be the one that is either front buffer or the back-buffer (a flip target). It can be only programmed once per context. **Programming Notes** It must be programmed before any draw call binding that render target base address.

FD

FD - FD

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	000C4h

Functional Disable. This register is used by SBIOS, not by driver.

DWord	Bit	Description
0	31:1	**RESERVED** Default Value: 00000000h Access: RO Reserved
	0	**FUNCTION_DISABLE** Default Value: 0b Access: R/W **Description** FD: 0 : Default - normal operation. 1 : When set, the function is disabled (configuration space is disabled). All new requests on the IOSF Primary bus, including any new configuration cycle requests are not claimed on IOSF Primary. This bit as no effect register accessibility via IOSF SB. Once programmed to '1', the only way to re-enable device 2 is via an IOSF SB write of '0' to this register. CHV, BSW: Wire is sent to PSF for decode purposes gvd_psf_dev2disable_nczfwoh.

FF Performance

FF_PERF - FF Performance

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	06B1Ch
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:16	**Mask** Project: All Access: WO Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:11	**Reserved** Project: All Access: r/w Format: PBC
	10:8	**Throttle counter value** Project: CHV, BSW Access: r/w Format: Disable Counter value defining how many clocks the interface needs to be slowed down. Value / Name / Description 0h / **[Default]** / Masked by default.
	7:3	**Reserved** Project: CHV, BSW Access: r/w Format: PBC

FF_PERF - FF Performance

	2	**Enable throttling for SF-WM interface** Access: r/w Format: Disable **Value / Name / Description** 0h / Disable / No throttling 1h / Enable / Enable throttling in all SF-WM interfaces
	1	**Enable throttling for SF-SBE interface** Access: r/w Format: Disable **Value / Name / Description** 0h / Disable / No throttling 1h / Enable / Enable throttling in all SF-SBE interfaces
	0	**Enable throttling for CL-SF interface** Access: r/w Format: Disable **Value / Name / Description** 0h / Disable / No throttling 1h / Enable / Enable throttling in all CL-SF interfaces

First Buffer Size and Start

FBSS - First Buffer Size and Start

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B420h

LPFCREG02 - First Buffer Size and Start

DWord	Bit	Description
0	31:16	**First Virtual Buffer Base** Access: R/W First Virtual Buffer Base: Programmed by driver to allocate a memory space for performance data storage. The buffer size should be aligned to the size of the memory allocated so it naturally aligns to the base (i.e. for 128KB bit[16]=0, 256KB bit[17:16]=0, 512KB bit[18:16]=0). Signal - lpconf_lpfc_virtual_base0 [31:16].
	15:12	**First Buffer Size** Access: R/W First Buffer Size: Determines the allowed buffer size for performance data storage. 0000b: 64KB. 0001b: 128KB. 0010b: 256KB. 0011b: 512KB. ... 1111b: 2GB. Signal - lpconf_lpfc_buffer_size0 [3:0].
	11:3	**Reserved** Project: CHV, BSW Access: RO Reserved.

FBSS - First Buffer Size and Start

	2	**Frame count and Draw call enable** Access: R/W Enables the replacement of a specific L3 performance counter value in the reported data with a 16-bit tag created from the concatenation of the "Frame Count" and "Draw Call Number" programmable bitfields in the "Frame Count and Draw Call Number" register. The exact counter replaced is dependent on the programmed value of the "Counter Enabling Selection" bitfield. The replaced counter is always the last one, except in the case only a single performance counter is enabled for reporting (in which no replacement occurs): **CNTRENSEL Value** / **Replaced Event Counter** 00 / No Replacement 01 / Counter 1 10 / Counter 3 11 / Counter 7
	1	**Reserved**
	0	**Master Counter Enable** Access: R/W Master Counter Enable: This is the global enable for performance tracking. Once set, it kicks off all performance tracking mechanism. Signal - lpconf_lpfc_master_cnt_en. This bit is used by all slices.

Flexible EU Event Control 0

EU_PERF_CNT_CTL0 - Flexible EU Event Control 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E458h

This register configures flexible EU event 0/1. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 1** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 1. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 1** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 1. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 1** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 1.
	11:8	**Fine Event Filter Select EU event 0** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 0. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 0**

EU_PERF_CNT_CTL0 - Flexible EU Event Control 0

		Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 0. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 0** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 0.

Flexible EU Event Control 1

EU_PERF_CNT_CTL1 - Flexible EU Event Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E558h

This register configures flexible EU event 2/3. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 3** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 3. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 3** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 3. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 3** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 3.
	11:8	**Fine Event Filter Select EU event 2** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 2. Note that the fine event filter is logically applied after the coarse event filter.

EU_PERF_CNT_CTL1 - Flexible EU Event Control 1

	7:4	**Coarse Event Filter Select EU event 2** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 2. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 2** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 2.

Flexible EU Event Control 2

EU_PERF_CNT_CTL2 - Flexible EU Event Control 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E658h

This register configures flexible EU event 4/5. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 5** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 5. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 5** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 5. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 5** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 5.
	11:8	**Fine Event Filter Select EU event 4** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 4. Note that the fine event filter is logically applied after the coarse event filter.

EU_PERF_CNT_CTL2 - Flexible EU Event Control 2

	7:4	**Coarse Event Filter Select EU event 4** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 4. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 4** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 4.

Flexible EU Event Control 3

EU_PERF_CNT_CTL3 - Flexible EU Event Control 3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E758h

This register configures flexible EU event 6/7. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 7** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 7. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 7** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 7. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 7** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 7.
	11:8	**Fine Event Filter Select EU event 6** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 6. Note that the fine event filter is logically applied after the coarse event filter.

EU_PERF_CNT_CTL3 - Flexible EU Event Control 3		
	7:4	**Coarse Event Filter Select EU event 6** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 6. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 6** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 6.

Flexible EU Event Control 4

EU_PERF_CNT_CTL4 - Flexible EU Event Control 4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E45Ch

This register configures flexible EU event 8/9. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 9** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 9. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 9** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 9. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 9** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 9.
	11:8	**Fine Event Filter Select EU event 8** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 8. Note that the fine event filter is logically applied after the coarse event filter.
	7:4	**Coarse Event Filter Select EU event 8**

EU_PERF_CNT_CTL4 - Flexible EU Event Control 4

		Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 8. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 8** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 8.

Flexible EU Event Control 5

EU_PERF_CNT_CTL5 - Flexible EU Event Control 5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E55Ch

This register configures flexible EU event 10/11. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 11** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 11. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 11** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 11. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 11** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 11.
	11:8	**Fine Event Filter Select EU event 10** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 10. Note that the fine event filter is logically applied after the coarse event filter.

EU_PERF_CNT_CTL5 - Flexible EU Event Control 5

	7:4	**Coarse Event Filter Select EU event 10** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 10. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 10** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 10.

Flexible EU Event Control 6

EU_PERF_CNT_CTL6 - Flexible EU Event Control 6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E65Ch

This register configures flexible EU event 12/13. Please refer to the description of the flexible EU events for more details on supported events. Please note that this register is render context saved/restored.

DWord	Bit	Description
0	31:24	**Reserved** Project: All Format: MBZ
	23:20	**Fine Event Filter Select EU event 13** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 13. Note that the fine event filter is logically applied after the coarse event filter.
	19:16	**Coarse Event Filter Select EU event 13** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 13. Note that the coarse event filter is logically applied before the fine event filter.
	15:12	**Increment Event for EU event 13** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 13.
	11:8	**Fine Event Filter Select EU event 12** Project: CHV, BSW Format: U4 This field controls which fine event filter is applied to the coarsely filtered increment event when creating flexible EU event 12. Note that the fine event filter is logically applied after the coarse event filter.

EU_PERF_CNT_CTL6 - Flexible EU Event Control 6

	7:4	**Coarse Event Filter Select EU event 12** Project: CHV, BSW Format: U4 This field controls which coarse event filter is applied to the selected increment event when creating flexible EU event 12. Note that the coarse event filter is logically applied before the fine event filter.
	3:0	**Increment Event for EU event 12** Project: CHV, BSW Format: U4 This field controls which increment event provides the basis for flexible EU event 12.

FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00002094
Access:	R/W
Size (in bits):	32
Address:	024D0h-024D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_RCSUNIT
Address:	024D4h-024D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_RCSUNIT
Address:	024D8h-024DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_RCSUNIT
Address:	024DCh-024DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_RCSUNIT
Address:	024E0h-024E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_RCSUNIT
Address:	024E4h-024E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_RCSUNIT
Address:	024E8h-024EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_RCSUNIT
Address:	024ECh-024EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_RCSUNIT
Address:	024F0h-024F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_RCSUNIT

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Address:	024F4h-024F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_RCSUNIT
Address:	024F8h-024FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_RCSUNIT
Address:	024FCh-024FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_RCSUNIT
Address:	124D0h-124D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VCSUNIT0
Address:	124D4h-124D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VCSUNIT0
Address:	124D8h-124DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VCSUNIT0
Address:	124DCh-124DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VCSUNIT0
Address:	124E0h-124E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VCSUNIT0
Address:	124E4h-124E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VCSUNIT0
Address:	124E8h-124EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VCSUNIT0
Address:	124ECh-124EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VCSUNIT0
Address:	124F0h-124F3h
Name:	FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
ShortName:	FORCE_TO_NONPRIV_8_VCSUNIT0
Address:	124F4h-124F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VCSUNIT0
Address:	124F8h-124FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VCSUNIT0
Address:	124FCh-124FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VCSUNIT0
Address:	1A4D0h-1A4D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VECSUNIT
Address:	1A4D4h-1A4D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VECSUNIT
Address:	1A4D8h-1A4DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VECSUNIT
Address:	1A4DCh-1A4DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VECSUNIT
Address:	1A4E0h-1A4E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VECSUNIT
Address:	1A4E4h-1A4E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VECSUNIT
Address:	1A4E8h-1A4EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VECSUNIT
Address:	1A4ECh-1A4EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VECSUNIT
Address:	1A4F0h-1A4F3h

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_VECSUNIT
Address:	1A4F4h-1A4F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VECSUNIT
Address:	1A4F8h-1A4FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VECSUNIT
Address:	1A4FCh-1A4FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VECSUNIT
Address:	1C4D0h-1C4D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_VCSUNIT1
Address:	1C4D4h-1C4D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_VCSUNIT1
Address:	1C4D8h-1C4DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_VCSUNIT1
Address:	1C4DCh-1C4DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_VCSUNIT1
Address:	1C4E0h-1C4E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_VCSUNIT1
Address:	1C4E4h-1C4E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_VCSUNIT1
Address:	1C4E8h-1C4EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_VCSUNIT1
Address:	1C4ECh-1C4EFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_7_VCSUNIT1

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV	
Address:	1C4F0h-1C4F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_VCSUNIT1
Address:	1C4F4h-1C4F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_VCSUNIT1
Address:	1C4F8h-1C4FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_VCSUNIT1
Address:	1C4FCh-1C4FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_VCSUNIT1
Address:	224D0h-224D3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_0_BCSUNIT
Address:	224D4h-224D7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_1_BCSUNIT
Address:	224D8h-224DBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_2_BCSUNIT
Address:	224DCh-224DFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_3_BCSUNIT
Address:	224E0h-224E3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_4_BCSUNIT
Address:	224E4h-224E7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_5_BCSUNIT
Address:	224E8h-224EBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_6_BCSUNIT
Address:	224ECh-224EFh
Name:	FORCE_TO_NONPRIV

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

ShortName:	FORCE_TO_NONPRIV_7_BCSUNIT
Address:	224F0h-224F3h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_8_BCSUNIT
Address:	224F4h-224F7h
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_9_BCSUNIT
Address:	224F8h-224FBh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_10_BCSUNIT
Address:	224FCh-224FFh
Name:	FORCE_TO_NONPRIV
ShortName:	FORCE_TO_NONPRIV_11_BCSUNIT

These registers are privilege registers and are not allowed to be written from non-privilege batch buffer. These are global registers and power context save/restored.

Programming Notes

RCS_FORCE_TO_NONPRIV registers in render CS must be used to force the below registers to be treated as non-privileged by HW:

- 0x7700 (GLOBAL_CLEAR_VALUE_0)
- 0x7704 (GLOBAL_CLEAR_VALUE_1)
- 0x7708 (GLOBAL_CLEAR_VALUE_2)
- 0x770C (GLOBAL_CLEAR_VALUE_3)

DWord	Bit	Description
0	31:26	**Reserved** Format: MBZ
	25:2	**Non Privilege Register Address** Format: MmioAddress[25:2] This field contains the MMIO offset of a register. MMIO offset programmed in this field will be treated as a non-privilege register by render command streamer while processing register writes from a non-privilege batch buffer. This register provides programmability is to extend the non-privilege register table mentioned in MI_BATCH_BUFFER_START command in render command streamer. Value: 825h — Name: **[Default]**
	1:0	**Reserved** Format: MBZ

Frame count and Draw call number

FCDCN - Frame count and Draw call number

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B430h

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:8	**Frame Number** Access: R/W Frame number is the first of two reporting tags that software (i.e. driver) may populate in order to provide reference points during L3 performance reporting modes. Should the "Frame Count and Draw Call Enable" bit (FCDCE) in the "First Buffer Size and Start" register be set, LPFC will selectively replace one of the reporting events with this programmable tag (in addition to the "Draw Call Number" field below). Software may use this to provide reference points for L3 performance counts when parsing the resulting data stream to align reported counts to higher-level operations. The original incarnation called for software to increment this value with each frame, however, the field is generic and may be used for any tagging purpose.
	7:0	**Draw call number** Access: R/W The draw call number is the second programmable reporting tag provided by LPFC. With this second programmable tag, a more granular sampling boundary may be created by software, or it may be used to provide an alternative reference point for tracking L3 performance. The original incarnation called for software to increment this value with every draw call, but the field is generic and may be used for any similar purpose.

FUSEWORD0

FUSEWORD0 - FUSEWORD0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182130h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW0** Default Value: 00000000h Access: RO Fuse Information DW0

FUSEWORD1

FUSEWORD1 - FUSEWORD1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182134h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW1** Default Value: 00000000h Access: RO Fuse Information DW1

FUSEWORD2

FUSEWORD2 - FUSEWORD2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182138h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW2** Default Value: 00000000h Access: RO Fuse Information DW2

FUSEWORD3

FUSEWORD3 - FUSEWORD3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18213Ch

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW3** Default Value: 00000000h Access: RO Fuse Information DW3

FUSEWORD4

FUSEWORD4 - FUSEWORD4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182140h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW4** Default Value: 00000000h Access: RO Fuse Information DW4

FUSEWORD5

FUSEWORD5 - FUSEWORD5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182144h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW5** Default Value: 00000000h Access: RO Fuse Information DW5

FUSEWORD6

FUSEWORD6 - FUSEWORD6	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182148h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW6** Default Value: 00000000h Access: RO Fuse Information DW6

FUSEWORD7

FUSEWORD7 - FUSEWORD7

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18214Ch

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW7** Default Value: 00000000h Access: RO Fuse Information DW7

FUSEWORD8

FUSEWORD8 - FUSEWORD8	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182150h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW8** Default Value: 00000000h Access: RO Fuse Information DW8

FUSEWORD9

FUSEWORD9 - FUSEWORD9

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182154h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW9** Default Value: 00000000h Access: RO Fuse Information DW9

FUSEWORD10

FUSEWORD10 - FUSEWORD10	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182158h

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW10** Default Value: 00000000h Access: RO Fuse information DW10

FUSEWORD11

FUSEWORD11 - FUSEWORD11	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18215Ch

Fuse readout information.

DWord	Bit	Description
0	31:0	**FUSEDW11** Default Value: 00000000h Access: RO Fuse information DW11

FUSEWORD12

FUSEWORD12 - FUSEWORD12	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182160h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW12** Default Value: 00000000h Access: RO Fuse information DW12

FUSEWORD13

FUSEWORD13 - FUSEWORD13

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182164h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW13** Default Value: 00000000h Access: RO Fuse information DW13

FUSEWORD14

FUSEWORD14 - FUSEWORD14	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182168h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW14** Default Value: 00000000h Access: RO Fuse information DW14 [31:28] - FUSE_GT_EU_DISABLE: SS1, ROW1 - EU[3:0] [27:24] - FUSE_GT_EU_DISABLE: SS1, ROW0 - EU[3:0] [23:20] - FUSE_GT_EU_DISABLE: SS0, ROW1 - EU[3:0] [19:16] - FUSE_GT_EU_DISABLE: SS0, ROW0 - EU[3:0] [11] - FUSE_GT_SUBSLICE_DISABLE - SS1 [10] - FUSE_GT_SUBSLICE_DISABLE - SS0

FUSEWORD15

FUSEWORD15 - FUSEWORD15

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18216Ch

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW15** Default Value: 00000000h Access: RO Fuse information DW15

FUSEWORD16

FUSEWORD16 - FUSEWORD16	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182170h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW16** Default Value: 00000000h Access: RO Fuse information DW16

FUSEWORD17

FUSEWORD17 - FUSEWORD17	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182174h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW17** Default Value: 00000000h Access: RO Spare

FUSEWORD18

FUSEWORD18 - FUSEWORD18	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182178h

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW18** Default Value: 00000000h Access: RO Spare

FUSEWORD19

FUSEWORD19 - FUSEWORD19	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18217Ch

Fuse readout information. CHV, BSW: New for CHV, BSW.

DWord	Bit	Description
0	31:0	**FUSEDW19** Default Value: 00000000h Access: RO Spare

G3D Control Register

G3DCTL - G3D Control Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	09024h	
G3D unit Control Register. Controls GUNIT's response to GAM Traffic.		

DWord	Bit	Description
0	31:27	**G3D Spare** Access: R/W Spare config bits sent to G3D.
	26	**RTN Mode** Access: R/W RTN mode. When set to one all GAM read returns are considered slow (no b2b to any source ID).
	25	**Return Ordering Queue Disable** Access: R/W ROQ Disable. When set to a '1', the G3D Return Ordering queue will be mostly disabled by forcing the Data queue to be considered full when it has one entry occupied.
	24	**Command Bandwidth Arb Mode** Access: R/W Command Bandwidth Arbitration mode. When programmed to a '1', reads will be allowed to go 1 cycle after writes as long as writes are not in write grant mode.
	23:0	**RSVD** Access: RO Reserved.

GAB Arbitration Programmable

GAB_AP - GAB Arbitration Programmable	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040F0h

DWord	Bit	Description
0	31:0	**Reserved**

GAB GAC GAM Idle

GABGACGAMIDLE - GAB GAC GAM Idle	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A540h-0A543h

DWord	Bit	Description
0	31:26	**Reserved** Access: RO
	25:16	**GAM Idle Timeout** Access: R/W **Programming Notes** This mode is not supported. It must be 0 at all times.
	15:10	**Reserved** Access: RO
	9:0	**Min GAB GAC Idle** Access: R/W **Programming Notes** This mode is not supported. It must be 0 at all times.

GAB LRA 0

GAB_LRA_0 - GAB LRA 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x1F100F00
Size (in bits):	32
Address:	04A70h

DWord	Bit	Description
0	31:29	**Reserved** Default Value: 000b Access: RO
	28:24	**GAB LRA1 Max** Default Value: 11111b Access: R/W Maximum value of programmable LRA1.
	23:21	**Reserved** Default Value: 000b Access: RO
	20:16	**GAB LRA1 Min** Default Value: 10000b Access: R/W Minimum value of programmable LRA1.
	15:13	**Reserved** Default Value: 000b Access: RO
	12:8	**GAB LRA0 Max** Default Value: 01111b Access: R/W Maximum value of programmable LRA0.
	7:5	**Reserved** Default Value: 000b Access: RO

GAB_LRA_0 - GAB LRA 0		
	4:0	**GABLRA0 Min** Default Value: 00000b Access: R/W Minimum value of programmable LRA0.

GAB LRA 1

GAB_LRA_1 - GAB LRA 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001
Size (in bits):	32
Address:	04A74h

DWord	Bit	Description
0	31:4	**Reserved** Default Value: 0000000h Access: RO
	3:2	**BLB** Default Value: 00b Access: R/W Which LRA should BLB use.
	1:0	**BCS** Default Value: 01b Access: R/W Which LRA should BCS use.

GAB unit Control Register

GAB_CTL_REG - GAB unit Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x000000BF
Access:	R/W
Size (in bits):	32
Address:	24000h

DefaultValue=FF0000BFh Trusted Type = 1

DWord	Bit	Description
0	31:9	**Reserved**
	8	**Continue after Page Fault** Value / Name / Description: 1 / GAB Set / Ipon receiving a page fault when requesting an address translation, GAB will set address bit 39 to 1 and continue. 0 / GAB Hang / GAB will hang on a page fault. Default = b0.
	7:6	**PPGTT BCS TLB LRA MIN** Default Value: 10b TLB Depth Partitioning Register In PP GTT Mode.
	5:4	**GAB write request priority signal value used in GAC arbitration** Default Value: 11b
	3:2	**GAB read only request priority signal value used in GAC arbitration** Default Value: 11b
	1:0	**GAB read request priority signal value used in GAC arbitration** Default Value: 11b

GAC_GAM Arbitration Counters Register 0

ARB_GAC_GAM_REQCNTS0 - GAC_GAM Arbitration Counters Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043A8h

DWord	Bit	Description
0	31:22	**Reserved**
	21:16	**Number of GAC WR requests to be accumulated before applying the arbitration**
	15:14	**Reserved**
	13:8	**Number of GAC R requests to be accumulated before applying the arbitration**
	7:6	**Reserved**
	5:0	**Number of GAC RO requests to be accumulated before applying the arbitration**

GAC_GAM Arbitration Counters Register 1

ARB_GAC_GAM_REQCNTS1 - GAC_GAM Arbitration Counters Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043ACh

DWord	Bit	Description
0	31:22	**Reserved**
	21:16	**Number of GAC WR requests to be accumulated before applying the arbitration**
	15:14	**Reserved**
	13:8	**Number of GAC R requests to be accumulated before applying the arbitration**
	7:6	**Reserved**
	5:0	**Number of GAC RO requests to be accumulated before applying the arbitration**

GAC_GAM R Arbitration Register 0

ARB_R_GAC_GAM0 - GAC_GAM R Arbitration Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043E0h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 0**
	11:9	**Goto field for entry 0 when request vector is 11b**
	8:6	**Goto field for entry 0 when request vector is 10b**
	5:3	**Goto field for entry 0 when request vector is 01b**
	2:0	**Goto field for entry 0 when request vector is 00b**

GAC_GAM R Arbitration Register 1

ARB_R_GAC_GAM1 - GAC_GAM R Arbitration Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043E4h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM R Arbitration Register 2

ARB_R_GAC_GAM2 - GAC_GAM R Arbitration Register 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043E8h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM R Arbitration Register 3

ARB_R_GAC_GAM3 - GAC_GAM R Arbitration Register 3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043ECh

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAC_GAM RO Arbitration Register 0

ARB_RO_GAC_GAM0 - GAC_GAM RO Arbitration Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043D0h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 01**
	11:9	**Goto field for entry 01 when request vector is 11b**
	8:6	**Goto field for entry 01 when request vector is 10b**
	5:3	**Goto field for entry 01 when request vector is 01b**
	2:0	**Goto field for entry 01 when request vector is 00b**

GAC_GAM RO Arbitration Register 1

ARB_RO_GAC_GAM1 - GAC_GAM RO Arbitration Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043D4h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM RO Arbitration Register 2

ARB_RO_GAC_GAM2 - GAC_GAM RO Arbitration Register 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043D8h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM RO Arbitration Register 3

ARB_RO_GAC_GAM3 - GAC_GAM RO Arbitration Register 3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043DCh

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAC_GAM WR Arbitration Register 0

ARB_WR_GAC_GAM0 - GAC_GAM WR Arbitration Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043F0h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 1**
	26:24	**Goto field for entry 1 when request vector is 11b**
	23:21	**Goto field for entry 1 when request vector is 10b**
	20:18	**Goto field for entry 1 when request vector is 01b**
	17:15	**Goto field for entry 1 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 0**
	11:9	**Goto field for entry 0 when request vector is 11b**
	8:6	**Goto field for entry 0 when request vector is 10b**
	5:3	**Goto field for entry 0 when request vector is 01b**
	2:0	**Goto field for entry 0 when request vector is 00b**

GAC_GAM WR Arbitration Register 1

ARB_WR_GAC_GAM1 - GAC_GAM WR Arbitration Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043F4h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 3**
	26:24	**Goto field for entry 3 when request vector is 11b**
	23:21	**Goto field for entry 3 when request vector is 10b**
	20:18	**Goto field for entry 3 when request vector is 01b**
	17:15	**Goto field for entry 3 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 2**
	11:9	**Goto field for entry 2 when request vector is 11b**
	8:6	**Goto field for entry 2 when request vector is 10b**
	5:3	**Goto field for entry 2 when request vector is 01b**
	2:0	**Goto field for entry 2 when request vector is 00b**

GAC_GAM WR Arbitration Register 2

ARB_WR_GAC_GAM2 - GAC_GAM WR Arbitration Register 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043F8h

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 5**
	26:24	**Goto field for entry 5 when request vector is 11b**
	23:21	**Goto field for entry 5 when request vector is 10b**
	20:18	**Goto field for entry 5 when request vector is 01b**
	17:15	**Goto field for entry 5 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 4**
	11:9	**Goto field for entry 4 when request vector is 11b**
	8:6	**Goto field for entry 4 when request vector is 10b**
	5:3	**Goto field for entry 4 when request vector is 01b**
	2:0	**Goto field for entry 4 when request vector is 00b**

GAC_GAM WR Arbitration Register 3

ARB_WR_GAC_GAM3 - GAC_GAM WR Arbitration Register 3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	043FCh

DWord	Bit	Description
0	31:28	**Reserved**
	27	**Priority for entry 7**
	26:24	**Goto field for entry 7 when request vector is 11b**
	23:21	**Goto field for entry 7 when request vector is 10b**
	20:18	**Goto field for entry 7 when request vector is 01b**
	17:15	**Goto field for entry 7 when request vector is 00b**
	14:13	**Reserved**
	12	**Priority for entry 6**
	11:9	**Goto field for entry 6 when request vector is 11b**
	8:6	**Goto field for entry 6 when request vector is 10b**
	5:3	**Goto field for entry 6 when request vector is 01b**
	2:0	**Goto field for entry 6 when request vector is 00b**

GAM and SA Communication Register

GAMSACOMREG - GAM and SA Communication Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	042A0h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**GAM and SA Communication Register 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**GAM and SA Communication Register 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**GAM and SA Communication Register 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**GAM and SA Communication Register 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**GAM and SA Communication Register 11** Default Value: 0b Access: R/W For Future Use.

GAMSACOMREG - GAM and SA Communication Register

		This bit is self clear.
	10	**GAM and SA Communication Register 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**GAM and SA Communication Register 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**GAM and SA Communication Register 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**GAM and SA Communication Register 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**GAM and SA Communication Register 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**GAM and SA Communication Register 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**GAM and SA Communication Register 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**GAM and SA Communication Register 3** Default Value: 0b Access: R/W

GAMSACOMREG - GAM and SA Communication Register

		For Future Use. This bit is self clear.
	2	**GAM and SA Communication Register 2** Default Value: 0b Access: R/W Bit2 - Root Table Address Update Request. This bit is self clear.
	1	**GAM and SA Communication Register 1** Default Value: 0b Access: R/W Bit1 - Queued Descriptor Request. This bit is self clear.
	0	**GAM and SA Communication Register 0** Default Value: 0b Access: R/W Bit0 - Context Cache Invalidator Request. This bit is self clear.

Gam Fub Done1 Lookup Register

DONE1_REG - Gam Fub Done1 Lookup Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0407Ch	
DWord	**Bit**	**Description**
0	31:0	**Gam Fub Done1 Lookup Reg** Default Value: 00000000h Access: RO GAM Done1 signals.

Gam Fub Done Lookup Register

DONE_REG - Gam Fub Done Lookup Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040B0h

DWord	Bit	Description
0	31:0	**Gam Fub Done Lookup Reg** Default Value: 00000000h Access: RO 31 CVS Credit Fifo is empty. 30 CVS TLB does not have any cycles. 29 Z Credit fifo is empty. 28 ZTLB does not have any cycles. 27 RCC Credit Fifo is empty. 26 RCC TLB does not have any cycles. 25 L3 Credit fifo is empty. 24 L3 TLB does not have any cycles. 23 VLF Credit fifo is empty. 22 VLF TLB does not have any cycles. 21 CASC Credit fifo empty. 20 CASC TLB does not have any cycles. 19 Miss Fub Done. 18 Read Stream Done. 17 Read Steam Fifo is empty. 16 Recycle Fifo in rstrm is empty. 15 TLB Pend Done. 14 TLB Pend PQ Array is done. 13 TLB pend PB Array is done. 12 Read route fub is done. 11 Gafm Data fifo is empty. 10 GAP data fifo is empty. 9 GAC data fifo is empty. 8 Wrdp is done with all the cycles. 7 Wrdp RID fifo is empty. 6 No hold from midarb to RTSTRM. 5 No hold from TLBPEND to MIDARB.

DONE_REG - Gam Fub Done Lookup Register		
		3 Tied to "1" - to be defined. 2 Fence FSM are IDLE. 1 Non PD Load Done. 0 Tied to "1" - to be defined.

GAM Put Delay

GAM_PUT_DLY - GAM Put Delay			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0401Ch		
Number of clocks to wait between puts			
DWord	**Bit**	**Description**	
0	31:0	**GAM PUT DELAY**	
		Default Value:	00000000h
		Access:	R/W

GAMT_DONE Register

GAMT_DONE - GAMT_DONE Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04AC0h

DWord	Bit	Description
0	31:0	**GAMT_DONE Register** Default Value: 00000000h Access: RO 31: vebxtlb_all_done_f 30: cvstlb_all_done_f 29: ztlb_all_done_f 28: l3tlb_all_done_f 27: rcctlb_all_done_f 26: mfxtlb_all_done_f 25: vlftlb_all_done_f 24: bwgtlb_all_done_f 23: gamwrrb_all_done_f 22: mfxsl1tlb_all_done_f 21: vlfsl1tlb_all_done_f 20: bwgtlb_fifo_empty 19: l3tlb_fifo_empty 18: ztlb_fifo_empty 17: rcctlb_fifo_empty 16: cvstlb_fifo_empty 15: vebxtlb_fifo_empty 14: mfxtlb_fifo_empty 13: mfxsl1tlb_fifo_empty 12: vlfsl1tlb_fifo_empty 11: vlftlb_fifo_empty 10: wrdp_gafm_fifo_empty 9: wrdp_gap_fifo_empty 8: wrdp_gacfg_fifo_empty 7: wrdp_cs_fifo_empty 6: wrdp_vecs_fifo_empty 5: wrdp_oacs_fifo_empty 4: wrdp_gacv_fifo_empty

GAMT_DONE - GAMT_DONE Register		
		3: Tied to 1 2: Tied to 1 1: Tied to 1 0: Tied to 1

GAMT_ECO_REG_RO_IA

GAMT_ECO_REG_RO_IA - GAMT_ECO_REG_RO_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04AB4h

DWord	Bit	Description
0	31:0	**GAMTECO_REG_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on DEV reset.

GAMT_ECO_REG_RW_IA

GAMT_ECO_REG_RW_IA - GAMT_ECO_REG_RW_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW, :GT2:B
Source:	PRM
Default Value:	0x0000AB1B
Size (in bits):	32
Address:	04AB0h

Programmable Request Count - VEBX and BLT

DWord	Bit	Description
0	31:0	**GAMTECO_REG_RW_IA** Default Value: 0000AB1Bh Access: R/W Bit[31:16] = Reserved. Bit[15:8] = Number of max outstanding cycles (Misses and Hits not present) that can be allowed to potentially fault = 171. Bit[7:6] = Reserved. Bit[5:0] = Number of max outstanding misses that can be allowed to potentially fault = 27.

GAMT Arbiter Mode Control

GAMTARBMODE - GAMT Arbiter Mode Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	04A08h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**GAMT Arbiter Mode Control 15** Default Value: 0b Access: R/W For Future Use
	14	**GAMT Arbiter Mode Control 14** Default Value: 0b Project: CHV, BSW Access: R/W 0 - Cache the TLB even if there is a FAULT in GAMW read return. 1 - Don't Cache the TLB if there is a fault in GAMW return.
	13	**GAMT Arbiter Mode Control 13** Default Value: 0b Access: R/W 0 - VEBXTLB clock gate enabled. 1 - VEBXTLB clock gate disabled.
	12	**GAMT Arbiter Mode Control 12** Default Value: 0b Access: R/W 0 - MFXSL1TLB clock gate enabled. 1 - MFXSL1TLB clock gate disabled.
	11	**GAMT Arbiter Mode Control 11** Default Value: 0b Access: R/W 0 - VLFSL1TLB clock gate enabled. 1 - VLFSL1TLB clock gate disabled.

GAMTARBMODE - GAMT Arbiter Mode Control

	10	**GAMT Arbiter Mode Control 10** Default Value: 0b Access: R/W 0 - gamwrrb clock gate enabled. 1 - gamwrrb clock gate disabled.
	9	**GAMT Arbiter Mode Control 9** Default Value: 0b Access: R/W 0 - BWGTLB clock gate enabled. 1 - BWGTLB clock gate disabled.
	8	**GAMT Arbiter Mode Control 8** Default Value: 0b Access: R/W 0 - VLFTLB clock gate enabled. 1 - VLFTLB clock gate disabled.
	7	**GAMT Arbiter Mode Control 7** Default Value: 0b Access: R/W 0 - MFXTLB clock gate enabled. 1 - MFXTLB clock gate disabled.
	6	**GAMT Arbiter Mode Control 6** Default Value: 0b Access: R/W 0 - RCCTLB clock gate enabled. 1 - RCCTLB clock gate disabled.
	5	**GAMT Arbiter Mode Control 5** Default Value: 0b Access: R/W 0 - L3TLB clock gate enabled. 1 - L3TLB clock gate disabled. bit[5] needs to be set as a work-around due to recent gacb bug. To update bit 5, a value of 0x00200020 needs to be written.
	4	**GAMT Arbiter Mode Control 4** Default Value: 0b Access: R/W 0 - ZTLB clock gate enabled. 1 - ZTLB clock gate disabled.

GAMTARBMODE - GAMT Arbiter Mode Control

	3	**GAMT Arbiter Mode Control 3** Default Value: 0b Access: R/W 0 - CVS clock gate enabled. 1 - CVS clock gate disabled.
	2	**GAMT Arbiter Mode Control 2** Default Value: 0b Access: R/W 0 - No reg_hdc_inval_ack_force - take the value from client. 1 - reg_hdc_inval_ack_force - force value to 1 - disregard client value.
	1	**GAMT Arbiter Mode Control 1** Default Value: 0b Access: R/W Bit [1]: Address Swizzling for Tiled Surfaces. This register location is updated via GFX Driver prior to enabling DRAM accesses. Driver needs to obtain the need for memory address swizzling via DRAM configuration registers and set the following bits (in Display Engine and Render/Media access streams). 0: No address Swizzling. 1: Address bit[1] needs to be swizzled for tiled surfaces.
	0	**GAMT Arbiter Mode Control 0** Default Value: 0b Access: R/W Bit[0]: GAM to Bypass GTT Translation. GAM to Bypass GTT Translation and pass logical addresses through with 0's padded on the MSBs to form the Physical Address.

GAMW_ECO_BUS_RO_IA

GAMW_ECO_BUS_RO_IA - GAMW_ECO_BUS_RO_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0408Ch

DWord	Bit	Description
0	31:0	**GAMWECO_BUS_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on BUS reset.

GAMW_ECO_BUS_RW_IA

GAMW_ECO_BUS_RW_IA - GAMW_ECO_BUS_RW_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04084h

DWord	Bit	Description
0	31:0	**GAMWECO_BUS_RW_IA** Default Value: 00000000h Access: R/W This register is for ECO usage. RW register with IA Access Type on BUS reset.

GAMW_ECO_DEV_RO_IA

GAMW_ECO_DEV_RO_IA - GAMW_ECO_DEV_RO_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04088h

DWord	Bit	Description
0	31:0	**GAMWECO_DEV_RO_IA** Default Value: 00000000h Access: RO This register is for ECO usage. RO register with IA Access Type on DEV reset.

GAMW_ECO_DEV_RW_IA

GAMW_ECO_DEV_RW_IA - GAMW_ECO_DEV_RW_IA	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04080h

DWord	Bit	Description
0	31:0	**GAMWECO_DEV_RW_IA** Default Value: 00000000h Access: R/W

GAMW Power Context Save

PWRCTXSAVE - GAMW Power Context Save

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04000h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**Extra Bits15** Default Value: 0b Access: R/W Extra Bits for future use.
	14	**Extra Bits14** Default Value: 0b Access: R/W Extra Bits for future use.
	13	**Extra Bits13** Default Value: 0b Access: R/W Extra Bits for future use.
	12	**Extra Bits12** Default Value: 0b Access: R/W Extra Bits for future use.
	11	**Extra Bits11** Default Value: 0b Access: R/W Extra Bits for future use.
	10	**Extra Bits10** Default Value: 0b Access: R/W Extra Bits for future use.

PWRCTXSAVE - GAMW Power Context Save

	9	**Power Context Save Request** Default Value: 0b Access: R/W Power Context Save Bit[9] Power Context Save Request 1'b0: Power context save is not being requested (default). 1'b1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. This bit is self clear.
	8:0	**Power Context Save Quad Word Credits** Default Value: 000000000b Access: R/W Power Context Save Bits[8:0] QWord Credits for Power Context Save Request An initial length packet is required per power context save session, but that packet does not consume a credit. See protocol description for more details. Minimum Credits = 1: Unit may send 1 QWord pair. Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Only valid with PWRCTX_SAVE_REQ (Bit9).

GARB Messaging Register for Boot Controller

MSG_GARB_MBC - GARB Messaging Register for Boot Controller

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	0801Ch

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:7	**Reserved** Access: RO
	6	**Fuse Download Done Indication** Access: R/W Fuse Download Done Indication 1'b0 : Fuse download is not complete yet <default> 1'b1 : Fuse download is complete gpmunit self-clears this bit upon sampling.
	5	**Boot Fetch Complete Indication** Access: R/W Boot Fetch Complete Indication 1'b0 : Boot Fetch is not complete yet <default> 1'b1 : Boot Fetch is complete gpmunit self-clears this bit upon sampling.
	4	**IDI Block Status** Access: R/W IDI Block Status 1'b0 : IDI interface is not blocked <default> 1'b1 : IDI interface is blocked
	3	**IDI Awake Status** Access: R/W IDI Awake Status 1'b0 : IDI interface is not ready <default> 1'b1 : IDI interface is ready
	2	**Credit Active Status** Access: R/W Credit Active Status 1'b0 : Send credit active deassert Event Bus Message on transition from 1'b1 => 1'b0 <default> 1'b1 : Send credit active assert Event Bus Message on transition from 1'b0 => 1'b1

MSG_GARB_MBC - GARB Messaging Register for Boot Controller

	1	**Global Arbitration Request** Access: R/W Global Arbitration Request 1'b0 : No request <default> 1'b1 : Request for arbitration Full handshake requiring ack
	0	**Busy Indication** Access: R/W Busy Indication 1'b0 : Idle <default> 1'b1 : Busy Full handshake requiring ack

GARB Messaging Register for Clocking Unit

MSG_GARB_GCP - GARB Messaging Register for Clocking Unit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08024h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:3	**Reserved** Access: RO
	2	**GCP Request to send FLR Complete Message to SA via GAM** Access: R/W GCP Request to send FLR Complete Message to SA via GAM 1'b0 : No request <default> 1'b1 : Send cycle on GA* Interface to address < address > with data < data > gpm will self-clear the request once it completes it. MBC needs to self-clear the acknowledgement once it sees it. GPM indicates a write cycle is complete once it puts it on the interface. GPM indicates a read cycle is complete once the read-return data comes back. gpmunit self-clears this bit upon sampling.
	1	**Global Arbitration Request** Access: R/W Global Arbitration Request 1'b0 : No request <default> 1'b1 : Request for arbitration Full handshake requiring ack
	0	**Busy Indication** Access: R/W Busy Indication 1'b0 : Idle <default> 1'b1 : Busy Full handshake requiring ack

Gather Constants Not Consumed By RCS

GATHER_CONST_PRODUCE_COUNT - Gather Constants Not Consumed By RCS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	0248Ch

This register keeps track of the outstanding Gather Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0	31:0	**Gather Constants Produce Count** This register keeps track of the outstanding Gather Constants produced by RS which are not yet consumed by Render Command Streamer. This register is part of the render context save and restore.

GDR Per Client Write Drop Enables

WR_DROP_MODE - GDR Per Client Write Drop Enables

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040B4h

DWord	Bit	Description
0	31:0	**GDR Per Client Write Drop Enables** Default Value: 00000000h Access: R/W 31 RSVD: Future use. 30 MBC write drop disable (0) or enable (1). 29 CS write drop disable (0) or enable (1). 28 SOL write drop disable (0) or enable (1). 27 RS write drop disable (0) or enable (1). 26 RCC write drop disable (0) or enable (1). 25 MSC write drop disable (0) or enable (1). 24 All L3 clients write drop disable (0) or enable (1). 23 STC write drop disable (0) or enable (1). 22 HIZ write drop disable (0) or enable (1). 21 RCZ write drop disable (0) or enable (1). 20 GAFS write drop disable (0) or enable (1). 19 GPM write drop disable (0) or enable (1). 18 GCP write drop disable (0) or enable (1). 17 VCS write drop disable (0) or enable (1). 16 BSP write drop disable (0) or enable (1). 15 VCR write drop disable (0) or enable (1). 14 VMX_RS write drop disable (0) or enable (1). 13 VMX_BS write drop disable (0) or enable (1). 12 VMX_RA write drop disable (0) or enable (1). 11 VMX_VDS write drop disable (0) or enable (1). 10 VLF_RS write drop disable (0) or enable (1). 9 VLF_FW write drop disable (0) or enable (1). 8 VECS write drop disable (0) or enable (1). 7 VEO write drop disable (0) or enable (1).

WR_DROP_MODE - GDR Per Client Write Drop Enables		
		5 uC (DMA) write drop disable (0) or enable (1). 4 BCS write drop disable (0) or enable (1). 3 BLB write drop disable (0) or enable (1). 2 W_BSP write drop disable (0) or enable (1). 1 W_VMX_RS write drop disable (0) or enable (1). 0 W_VMX_BS write drop disable (0) or enable (1).

GDR Write Drop

GDR_WR_DRP - GDR Write Drop	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04020h

DWord	Bit	Description
0	31:0	**GDR_WRITE_DROP** Default Value: 00000000h Access: R/W

General Purpose Power Management Performance Idle Hysteresis

GPMPIHYST - General Purpose Power Management Performance Idle Hysteresis

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A070h-0A073h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Performance Idle Hysteresis Direction** Access: R/W Idle intervals must be longer than this value to be considered idle. 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_idle_hyst[23:0] FYI: 0 means disabled.

GFX_FLSH_CNT

GFX_FLSH_CNT - GFX_FLSH_CNT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	101008h

Used to flush Gunit TLB

DWord	Bit	Description
0	31:1	**RESERVED** Default Value: 00000000h Access: RO Reserved
	0	**GfxFlshCntl** Default Value: 0b Access: WO Access type of this register is WO. A write to this bit flushes the Gfx TLB in GUNIT. The data associated with the write is discarded and a read return all 0s.

GFX Arbiter Client Priority Control

GFX_PRIO_CTRL - GFX Arbiter Client Priority Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x880A2D10
Size (in bits):	32
Address:	04A00h

DWord	Bit	Description
0	31:27	**Read Rstrm Max Reject** Default Value: 10001b Access: R/W
	26:21	**Extra Bits** Default Value: 000000b Access: R/W
	20:18	**sol_gam_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [18] is NOT Used.
	17:15	**veo_gam_priority** Default Value: 100b Access: R/W Client Priority Control Bits - Lowest Bit [15] is NOT Used.
	14:12	**vfw_gam_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [12] is NOT Used.
	11:9	**gapc_gam_c_priority** Default Value: 110b Access: R/W Client Priority Control Bits - Lowest Bit [9] is NOT Used.
	8:6	**gapc_gam_z_priority** Default Value: 100b Access: R/W Client Priority Control Bits - Lowest Bit [6] is NOT Used.

GFX_PRIO_CTRL - GFX Arbiter Client Priority Control

	5:3	**gapc_gam_l3_priority** Default Value: 010b Access: R/W Client Priority Control Bits - Lowest Bit [3] is NOT Used.
	2:0	**csrsvf_gam_priority** Default Value: 000b Access: R/W Client Priority Control Bits - Lowest Bit [0] is NOT Used.

GFX Context Element Descriptor (High Part)

GFX_CTX_EDR_H - GFX Context Element Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04404h

DWord	Bit	Description
0	31:0	**GFX Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W Bit[63:32] - Context ID: Context identification number assigned to separate this context from others. Context IDs need to be recycled in such a way that there cannot be two active contexts with the same ID. This is a unique identification number by which a context is identified and referenced.

GFX Context Element Descriptor (Low Part)

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04400h

DWord	Bit	Description
0	31:0	**GFX Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W Bit[31:12] - LRCA: Command Streamer Only. Bit[8] - Privileged Context / GGTT vs PPGTT mode: Differs in legacy vs advanced context modes: In Legacy Context: Defines the page tables to be used. This is how page walker comes to know PPGTT vs GGTT selection for the entire context. 0: Use Global GTT. 1: Use Per-Process GTT. In Advanced Context: Defines the privilege level for the context. 0: User mode context. 1: Supervisor mode context. Bit[5] - Deeper IA coherency Support: In Advanced Context: Defines the level of IA coherency. 0: IA coherency is provided at LLC level for all streams of GPU (i.e. Gen7.5 like mode). 1: IA coherency is provided at L3 level for EU data accesses of GPU. Bit[4] - A and D Support / 32 and 64b Address Support: Differs in legacy vs advanced context modes: In Legacy Context: Defines 32b vs 64b (48b canonical) addressing format: 0: 32b addressing format. 1: 64b (48b canonical) addressing format. In Advanced Context: Defines A and D bit support: 0: A and D bit management in page tables is NOT supported. 1: A and D bit management in page tables is supported. Bit[3] - Context Type: Legacy vs Advanced: Defines the context type 0: Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models). Note that advanced context is not bounded to GPGPU. 1: Legacy Context: Defines the context as legacy mode which is similar to prior generations of Gen8. Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected.

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)

		Bit[2] - FR: Command streamer specific.
		Bit[1] - Scheduling Mode: 1: Indicates execlist mode of scheduling. 0: Indicates Ring Buffer mode of scheduling. Bit[0] - Valid: Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it continues but flags an error.

GFX Context Element Descriptor (Low Part)

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04400h

DWord	Bit	Description
0	31:0	**GFX Context Element Descriptor** Default Value: 00000009h Access: R/W Bit[31:12] - LRCA: Command Streamer Only. Bit[8] - Privileged Context / GGTT vs PPGTT mode: Differs in legacy vs advanced context modes: In Legacy Context: Defines the page tables to be used. This is how page walker comes to know PPGTT vs GGTT selection for the entire context. 0: Use Global GTT. 1: Use Per-Process GTT. In Advanced Context: Defines the privilege level for the context. 0: User mode context. 1: Supervisor mode context. Bit[5] - Deeper IA coherency Support: In Advanced Context: Defines the level of IA coherency. 0: IA coherency is provided at LLC level for all streams of GPU (i.e. Gen7.5 like mode). 1: IA coherency is provided at L3 level for EU data accesses of GPU. Bit[4] - A and D Support / 32 and 64b Address Support: Differs in legacy vs advanced context modes: In Legacy Context: Defines 32b vs 64b (48b canonical) addressing format: 0: 32b addressing format. 1: 64b (48b canonical) addressing format. In Advanced Context: Defines A and D bit support: 0: A and D bit management in page tables is NOT supported. 1: A and D bit management in page tables is supported. Bit[3] - Context Type: Legacy vs Advanced: Defines the context type 0: Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models). Note that advanced context is not bounded to GPGPU. 1: Legacy Context: Defines the context as legacy mode which is similar to prior generations of Gen8. Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected.

GFX_CTX_EDR_L - GFX Context Element Descriptor (Low Part)

		Bit[2] - FR: Command streamer specific.
		Bit[1] - Scheduling Mode: 1: Indicates execlist mode of scheduling. 0: Indicates Ring Buffer mode of scheduling. Bit[0] - Valid: Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it continues but flags an error.

GFX PDP0/PML4/PASID Descriptor (High Part)

GFX_CTX_PDP0_H - GFX PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0440Ch

DWord	Bit	Description
0	31:0	**GFX PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W PDP0/PML4/PASID: This register can contain three values which depend on the element descriptor definition. PASID[19:0]: Populated in the first 20bits of the register and selected when Advanced Context flag is set. PML4[38:12]: Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected. PDP0[38:12]: Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping. Note: This is a guest physical address.

GFX PDP0/PML4/PASID Descriptor (Low Part)

GFX_CTX_PDP0_L - GFX PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04408h

DWord	Bit	Description
0	31:0	**GFX PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W PDP0/PML4/PASID: This register can contain three values which depend on the element descriptor definition. PASID[19:0]: Populated in the first 20 bits of the register and selected when Advanced Context flag is set. PML4[38:12]: Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected. PDP0[38:12]: Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping. Note: This is a guest physical address.

GFX PDP1 Descriptor Register (High Part)

GFX_CTX_PDP1_H - GFX PDP1 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04414h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping. Note: This is a guest physical address.

GFX PDP1 Descriptor Register (Low Part)

GFX_CTX_PDP1_L - GFX PDP1 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04410h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping. Note: This is a guest physical address.

GFX PDP2 Descriptor Register (High Part)

GFX_CTX_PDP2_H - GFX PDP2 Descriptor Register (High Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0441Ch	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping. Note: This is a guest physical address.

GFX PDP2 Descriptor Register (Low Part)

GFX_CTX_PDP2_L - GFX PDP2 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04418h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping. Note: This is a guest physical address.

GFX PDP3 Descriptor Register (High Part)

GFX_CTX_PDP3_H - GFX PDP3 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04424h

DWord	Bit	Description
0	31:0	**GFX PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping. Note: This is a guest physical address.

GFX PDP3 Descriptor Register (Low Part)

GFX_CTX_PDP3_L - GFX PDP3 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04420h	
DWord	**Bit**	**Description**
0	31:0	**GFX PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping. Note: This is a guest physical address.

GGC

GGC - GGC

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000028
Size (in bits):	32
Address:	00050h

GMCH Graphics Control Register.

DWord	Bit	Description
0	31:15	**RESERVED** Default Value: 00000h Access: RO Reserved
	14	**VAMEN** Default Value: 0b Access: R/W Lock Enables the use of the iGFX engines for Versatile Acceleration. 1 - iGFX engines are in Versatile Acceleration Mode. Device 2 Class Code is 048000h. 0 - iGFX engines are in iGFX Mode. Device 2 Class Code is 030000h.
	13:10	**RESERVED** Default Value: 0h Access: RO Reserved
	9:8	**GGMS** Default Value: 00b Access: R/W Lock GTT Graphics Memory Size (GGMS): This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics Translation Table. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. GSM is assumed to be a contiguous physical DRAM space with DSM, and BIOS needs to allocate a contiguous memory chunk. Hardware will derive the base of GSM from DSM only using the GSM size programmed in the register. Hardware functionality in case of programming this value to Reserved is not guaranteed. 0h:No Preallocated Memory 1h: 2MB of Preallocated Memory 2h: 4MB of Preallocated Memory 3h: 8MB of Preallocated Memory

GGC - GGC

	7:3	**GMS** Default Value: 00101b Access: R/W Lock Graphics Mode Select(GMS): This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled. BIOS Requirement: BIOS must not set this field to 0h if IVD (bit 1 of this register) is 0. 0h = 0MB 1h = 32MB 2h = 64MB 3h = 96MB 4h = 128MB 5h = 160MB 6h = 192MB 7h = 224MB 8h = 256MB 9h = 288MB Ah = 320MB Bh = 352MB Ch = 384MB Dh = 416MB Eh = 448MB Fh = 480MB 10h = 512MB 11h = 8MB 12h = 12MB 13h = 16MB 14h = 20MB 15h = 24MB 16h = 28MB 17h = 36MB 18h = 40MB 19h = 44MB 1Ah = 48MB 1Bh = 52MB 1Ch = 56MB 1Dh = 60MB 1Eh = Reserved 1Fh = Reserved ..... 20h:1024MB (Not supported for CHV, BSW)

GGC - GGC

		..... 30h:1536MB (Not supported for CHV, BSW) ..... 40h:2048MB (Not supported for CHV, BSW) ..... 80h:4096MB (Not supported for CHV, BSW) 81h - FF:Reserved Other = Reserved When GMS != '0 (and VD=0): Address[31:0] is compared with VGA memory range. (The VGA memory range is A_0000h to B_FFFFh.). If there is a match and MSE = 1 and MEMRD or MEMWR, the access will route as a Rmdwvgamemen_cr cycle on the RMbus. If the RMbus returns a hit the GVD will select the command. As well, when 0 the GVD will check if scldown3_address[15:0] is one of the VGA IO register range. (The VGA IO range is 03B0h - 03BBh and 03C0h - 03DFh.) If there is a match and IOSE = 1 and the SCL command is either an IORD or IOWR, the GVD will intiate a (VGA) register cycle on the RMbus. If the RMbus returns a hit the GVD will select the command When GMS == '0 : No address compare will occur against VGA memory range or the VGA IO register range. Also, CC[15:8] is changed to 8'h80 from 8'h00
	2	**RESERVED** Default Value: 0b Access: RO Reserved
	1	**VGA_DISABLE** Default Value: 0b Access: R/W Lock VGA Disable (VD): 0: Enable. Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00. 1: Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub- Class Code field within Device 2 function 0 Class Code register is 80. BIOS Requirement: BIOS must not set this bit to 0 if the GMS field pre-allocates no memory. This bit MUST be set to 1 if Device 2 is disabled either via a fuse or fuse override (CAPID0[38] = 1) or via a register (DEVEN[3] = 0).
	0	**GGCLCK** Default Value: 0b Access: R/W Lock When set to 1b, this will lock all the bits in this register.

Global Clear Value Register 0

GLOBAL_CLEAR_VALUE_0 - Global Clear Value Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07700h

This register is to be used to program bits 31:0 of the Global clear value to be used for non MSRT render targets. This value will be used by pixel backend hardware in place of the 0/1 clear values, if the "Use Global Clear Value" bit is set.

DWord	Bit	Description
0	31:0	**Global Clear Value 0** Contains bits 31:0 of the global clear value. Clear value will be in native RT format. Clear Value for 8/16/32 bpp RTs will be contained in this format. For the rest it will contain bits 31:0.

Global Clear Value Register 1

GLOBAL_CLEAR_VALUE_1 - Global Clear Value Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07704h

This register is to be used to program bits 63:32 of the Global clear value to be used for non MSRT render targets. This value will be used by pixel backend hardware in place of the 0/1 clear values, if the "Use Global Clear Value" bit is set.

DWord	Bit	Description
0	31:0	**Global Clear Value 1** Contains bits 63:32 of the global clear value. Clear value will be in native RT format. This field will contain bits 63:32 of the clear value of 64 bpp and 128 bpp RTs.

Global Clear Value Register 2

GLOBAL_CLEAR_VALUE_2 - Global Clear Value Register 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	07708h

This register is to be used to program bits 95:64 of the Global clear value to be used for non MSRT render targets. This value will be used by pixel backend hardware in place of the 0/1 clear values, if the "Use Global Clear Value" bit is set.

DWord	Bit	Description
0	31:0	**Global Clear Value 2** Contains bits 95:64 of the global clear value. Clear value will be in native RT format. Clear value will be in native RT format. This field will contain bits 95:64 of the clear value for 128 bpp RTs.

Global Clear Value Register 3

GLOBAL_CLEAR_VALUE_3 - Global Clear Value Register 3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0770Ch

This register is to be used to program bits 127:96 of the Global clear value to be used for non MSRT render targets. This value will be used by pixel backend hardware in place of the 0/1 clear values, if the "Use Global Clear Value" bit is set.

DWord	Bit	Description
0	31:0	**Global Clear Value 3** Contains bits 127:96 of the global clear value. Clear value will be in native RT format. This field will contain bits 127:96 of the clear value for 128 bpp RTs.

Global Invalidation Register

GLBLINVL - Global Invalidation Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B404h	
DWord	**Bit**	**Description**
0	31:3	**Reserved** Access: RO Reserved.
	2	**Cross sync read disable** Access: R/W lpconf_crs_sync_dis: Cross Sync Read Disable (CSRD). Cross Sync Read Disable (CSRD): Cross Sync Read Disable: upon a SYNC from HDC, follow with a write to cross SYNC Push and read to the same address. When set read is disabled.
	1	**Disables hashing function** Access: R/W Disables hashing function (DISHHF): Disables hashing function to generate bank_id[1:0] for L3$ bank accessing, and forces the use of address[7:6] for bank_id[1:0].0: (default) Hash function enabled to generate L3$ bank IDs. 1: L3$ address[7:6] used as L3$ bank IDs. lpconf_csr_l3bankidhashdis. (This bit needs to set corresponding bit lncf_csr_l3bankidhashdis in LNCF.)
	0	**Reserved**

Global System Interrupt Routine

EU_GLOBAL_SIP - Global System Interrupt Routine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E42Ch

DWord	Bit	Description
0	31:3	**Global SIP** Format: GraphicsAddress[31:3] Specifies the base address for System Interrupt Routine that over-rides the SIP set by the state (STATE_SIP).
	2:1	**Reserved** Format: MBZ
	0	**Global SIP Enable** The bit specifies if the System Routine starts from the Global SIP provided by the DW OR the SIP provided by the state (STATE_SIP) Value: 0 — Name: SIP used is from STATE_EIP Value: 1 — Name: SIP used is from MMIO register

GMADR_LSB

GMADR_LSB - GMADR_LSB

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0000000C
Size (in bits):	32
Address:	00018h

Gfx Aperture location.
GMADR is a Prefetchable range in order to apply USWC attribute (from the processor point of view) to that range. The USWC attribute is used by the processor for write combining.
Accesses to this range will be translated to DRAM Physical memory addresses. Fence registers may be used to sub-divide this range and allow tiled surfaces (determined by fence registers).
The supported sizes are determined by the MSAC register.

DWord	Bit	Description
0	31	**ADMSK4096** Default Value: 0b Access: R/W Lock 4096MB Address Mask (ADMSK4096): Locked with MSAC.APSZ[4]. See MSAC (Dev2, Func 0, offset 62h) for details.
	30	**ADMSK2048** Default Value: 0b Access: R/W Lock 2048MB Address Mask (ADMSK2048): Locked with MSAC.APSZ[3]. See MSAC (Dev2, Func 0, offset 62h) for details.
	29	**ADMSK1024** Default Value: 0b Access: R/W Lock 1024MB Address Mask (ADMSK1024): Locked with MSAC.APSZ[2]. See MSAC (Dev2, Func 0, offset 62h) for details.
	28	**ADMSK512** Default Value: 0b Access: R/W Lock 512MB Address Mask (ADMSK512): Locked with MSAC.APSZ[1]. See MSAC (Dev2, Func 0, offset 62h) for details.

GMADR_LSB - GMADR_LSB

	27	**ADMSK256** Default Value: 0b Access: R/W Lock 256MB Address Mask (ADMSK256): Locked with MSAC.APSZ[0].See MSAC (Dev2, Func 0, offset 62h) for details.
	26:4	**Reserved** Default Value: 000000h Access: RO Address Mask (ADM): Hardwired to 0s to indicate at least 128MB address range.
	3	**PREFMEM** Default Value: 1b Access: RO Prefetchable Memory (PREFMEM): Hardwired to 1 to enable prefetching.
	2:1	**MEMTYP** Default Value: 10b Access: RO Memory Type (MEMTYP): 00: To indicate 32 bit base address 01: Reserved 10: To indicate 64 bit base address 11: Reserved
	0	**Reserved** Default Value: 0b Access: RO Memory/IO Space (MIOS): Hardwired to 0 to indicate memory space.

GMADR_MSB

GMADR_MSB - GMADR_MSB

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0001Ch

Gfx Aperture location.
GMADR is a Prefetchable range in order to apply USWC attribute (from the processor point of view) to that range. The USWC attribute is used by the processor for write combining.
Accesses to this range will be translated to DRAM Physical memory addresses. Fence registers may be used to sub-divide this range and allow tiled surfaces (determined by fence registers).
The supported sizes are determined by the MSAC register.

DWord	Bit	Description
0	31:4	**MBA_MSB28** Default Value: 0000000h Access: R/W Set by the OS, these bits correspond to address signals [63:39] — Project: Made them spare RW bits. — Project: CHV, BSW
	3:0	**MBA** Default Value: 0h Access: R/W Memory Base Address (MBA)Set by the OS, these bits correspond to address signals [35:32]

GMBC Message Register

GMBCMSG - GMBC Message Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	08500h

GMBC Message Register

DWord	Bit	Description
0	31:12	**RSVD** Access: RO Reserved
	11	**Context Restore Message** Access: R/W Context Restore Message 1 = Context Restore Message From CS 0 = Not a Message, GMBC clears when sending ACK to CS.
	10	**MBC BUSY ACK Message** Access: R/W 1 = Busy Ack message from GPMG 0 = IDLE (non-busy) Ack message from GPMG
	9:7	**RSVD** Access: RO Reserved
	6	**Block Message** Access: R/W Block Message from GPMG 1 = Block Outbound GAM Traffic and wait for outstanding GAM write requests to be GO'd and reads to return 0 = Unblock Outbound GAM traffic
	5	**RSVD** Access: RO Reserved
	4	**Arbitration request/release ACK** Access: R/W

GMBCMSG - GMBC Message Register

		Global Arb request ACK message: 1 = GPMG ACK of GMBC Global Arb Update Request. 0 = GPMG ACK of GMBC Global Arb Release Request.
	3	**RSVD** Access: RO Reserved
	2	**Fuse Fetch Message** Access: R/W Fuse Fetch message: written to 1 by GPMG to indicate GMBC should perform a fuse fetch. Cleared by GMBC when done with the fuse fetch.
	1:0	**RSVD** Access: RO Reserved

GO Messaging Register for GAMunit

MSG_GO_GAM - GO Messaging Register for GAMunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08028h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.
GA* Response to Allow Graphics Cycles to Read/Write from Memory
1'b0 : No gfx cycles allowed to memory <default>
1'b1 : Allow gfx cycles to memory
gpm currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.

DWord	Bit	Description
0	15:7	**Reserved** Access: RO
	6	**GA* Response to Allow Wi-Di Graphics Cycles to Read/Write from Memory** Access: R/W [6] Controls Wi-Di Cycles (winunit)
	5	**Reserved**
	4	**GA* Response to Allow Blitter Graphics Cycles to Read/Write from Memory** Access: R/W [4] Controls Blitter Cycles (bcsunit)
	3	**GA* Response to Allow VEBox Graphics Cycles to Read/Write from Memory** Access: R/W [3] Controls VEBox Cycles (vecsunit)
	2	**GA* Response to Allow Media1 Graphics Cycles to Read/Write from Memory** Access: R/W [2] Controls Media1 Cycles (vcs1unit)
	1	**GA* Response to Allow Media0 Graphics Cycles to Read/Write from Memory** Access: R/W [1] Controls Media0 Cycles (vcs0unit)

MSG_GO_GAM - GO Messaging Register for GAMunit

	0	**GA* Response to Allow Render Graphics Cycles to Read/Write from Memory** Access: R/W [0] Controls Render Cycles (csunit)

Go Protocol GAM Request

GO_GAM_REQ - Go Protocol GAM Request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	040D0h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Reserved.
	15	**GO_PROTOCOL_GAM_REQUEST15** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for FLR (device) reset (cdevrst_b).
	14	**GO_PROTOCOL_GAM_REQUEST14** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Media1 reset (vcs1unit).
	13	**GO_PROTOCOL_GAM_REQUEST13** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Wi-Di reset (winunit).

GO_GAM_REQ - Go Protocol GAM Request

12	**GO_PROTOCOL_GAM_REQUEST12** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset).
11	**GO_PROTOCOL_GAM_REQUEST11** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Blitter reset (bcsunit).
10	**GO_PROTOCOL_GAM_REQUEST10** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for VEBox reset (vecsunit).
9	**GO_PROTOCOL_GAM_REQUEST9** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Media0 reset (vcs0unit).
8	**GO_PROTOCOL_GAM_REQUEST8** Default Value: 0b Project: CHV, BSW Access: R/W Go Protocol Request Reasons: 1'b0: Engine will NOT be resetting. 1'b1: Engine will be resetting (FLR/RC6 Enter or Soft Reset). Preparation for Render reset (csunit).

GO_GAM_REQ - Go Protocol GAM Request

	7	**Reserved**
	6	**GO_PROTOCOL_GAM_REQUEST6** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Wi-Di Cycles (winunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	5	**Reserved**
	4	**GO_PROTOCOL_GAM_REQUEST4** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Blitter Cycles (bcsunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	3	**GO_PROTOCOL_GAM_REQUEST3** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls VEBox Cycles (vecsunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	2	**GO_PROTOCOL_GAM_REQUEST2** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Media1 Cycles (vcs1unit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	1	**GO_PROTOCOL_GAM_REQUEST1** Default Value: 0b Access: R/W

GO_GAM_REQ - Go Protocol GAM Request

		GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Media0 Cycles (vcs0unit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.
	0	**GO_PROTOCOL_GAM_REQUEST0** Default Value: 0b Access: R/W GPM to GAM Go Protocol Request. 0: No graphic cycles allowed to memory (default). 1: Allow graphic cycles to memory. Controls Render Cycles (csunit). GPM currently only ever sends the same GO request for all agents; either all is go=0, or all is go=1.

GPA to HPA Translation Request

GPA2HPAR - GPA to HPA Translation Request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0420Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**GPA to HPA Translation Request 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**GPA to HPA Translation Request 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**GPA to HPA Translation Request 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**GPA to HPA Translation Request 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**GPA to HPA Translation Request 11** Default Value: 0b Access: R/W For Future Use.

GPA2HPAR - GPA to HPA Translation Request

		This bit is self clear.
	10	**GPA to HPA Translation Request 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**GPA to HPA Translation Request 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**GPA to HPA Translation Request 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**GPA to HPA Translation Request 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**GPA to HPA Translation Request 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**GPA to HPA Translation Request 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**GPA to HPA Translation Request 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**GPA to HPA Translation Request 3** Default Value: 0b Access: R/W

GPA2HPAR - GPA to HPA Translation Request

		For Future Use. This bit is self clear.
	2	**GPA to HPA Translation Request 2** Default Value: 0b Access: R/W Bit[2]: A request for GPA to HPA translation. Note that GPA register should have been written prior to sending the message for the translation. Mask bit[18] needs to be enabled to program the register. This bit is self clear.
	1	**GPA to HPA Translation Request 1** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	0	**GPA to HPA Translation Request 0** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

GPA value for GPA to HPA Translation

GPA2HPAV - GPA value for GPA to HPA Translation		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04210h	
DWord	**Bit**	**Description**
0	31:0	**GPA value for GPA to HPA Translation** Default Value: 00000000h Access: R/W The GPA value of the page that requires the GPA=>HPA translation bits[39:12] map to [28:1] of the register.

GPGPU Context Restore Request To TDL

GPGPU_CTX_RESTORE - GPGPU Context Restore Request To TDL	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S0_SS0
Valid Projects:	CHV, BSW
Address:	0E5CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S0_SS1
Valid Projects:	CHV, BSW
Address:	0E6CCh
Name:	GPGPU Context Restore Request To TDL Slice 0 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S0_SS2
Valid Projects:	CHV, BSW
Address:	0E4DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S1_SS0
Valid Projects:	CHV, BSW
Address:	0E5DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S1_SS1
Valid Projects:	CHV, BSW
Address:	0E6DCh
Name:	GPGPU Context Restore Request To TDL Slice 1 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S1_SS2
Valid Projects:	CHV, BSW
Address:	0E4ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 0
ShortName:	GPGPU_CTX_RESTORE_S2_SS0
Valid Projects:	CHV, BSW

GPGPU_CTX_RESTORE - GPGPU Context Restore Request To TDL

Address:	0E5ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 1
ShortName:	GPGPU_CTX_RESTORE_S2_SS1
Valid Projects:	CHV, BSW

Address:	0E6ECh
Name:	GPGPU Context Restore Request To TDL Slice 2 SubSlice 2
ShortName:	GPGPU_CTX_RESTORE_S2_SS2
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

GPGPU Context Save Request To TDL

GPGPU_CTX_SAVE - GPGPU Context Save Request To TDL	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E4D8h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

GPGPU Dispatch Dimension X

GPGPU_DISPATCHDIMX - GPGPU Dispatch Dimension X

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02500h

DWord	Bit	Description
0	31:0	**Dispatch Dimension X** Format: U32 The number of thread groups to be dispatched in the X dimension (max x + 1). Value: 0, FFFFFFFFh — Name: — Project: CHV, BSW

GPGPU Dispatch Dimension Y

GPGPU_DISPATCHDIMY - GPGPU Dispatch Dimension Y

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02504h

DWord	Bit	Description
0	31:0	**Dispatch Dimension Y** Format: U32 The number of thread groups to be dispatched in the Y dimension (max y + 1 Value: 0, FFFFFFFFh; Name: ; Project: CHV, BSW

GPGPU Dispatch Dimension Z

GPGPU_DISPATCHDIMZ - GPGPU Dispatch Dimension Z

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02508h

DWord	Bit	Description
0	31:0	**Dispatch Dimension Z** Format: U32 The number of thread groups to be dispatched in the Zdimension (max Z + 1) Value: 0, FFFFFFFFh; Name: ; Project: CHV, BSW

GPU_Ticks_Counter

GPU_TICKS - GPU_Ticks_Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02910h
Valid Projects:	CHV, BSW

Reading this register returns the live value of the GPU ticks counter that is sampled and included in the performance counter report header.

DWord	Bit	Description
0	31:0	**Considerations** Format: U32 This 32-bit field returns bits 31:0 of the GPU tick counter value when read. Note that there is no "latch and hold" mechanism for performance counters when they are accessed through MMIO, so the value returned from this register may be different on back-to-back reads.

Graphics Device Reset Control

GDRST - Graphics Device Reset Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0941Ch

Graphics Device Reset Control Registers

DWord	Bit	Description
0	31:7	**Reserved** Access: RO Reserved
	6	**Reserved**: MBZ
	5	**Reserved**
	4	**Initiate Graphics Vebox Soft Reset** Access: R/W Set Graphics VEbox Soft-Reset Control: '1' : Initiate a graphics Vebox domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	3	**Initiate Graphics Blitter Soft Reset** Access: R/W Set Graphics Blitter Soft-Reset Control: '1' : Initiate a graphics blitter domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	2	**Initiate Graphics Media Soft Reset** Access: R/W Set Graphics Media Soft-Reset Control: '1' : Initiate a graphics media 0 domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit.

GDRST - Graphics Device Reset Control

		Note: This is a non-posted register.
	1	**Initiate Graphics Render Soft Reset** Access: R/W Set Graphics Render Soft-Reset Control: '1' : Initiate a graphics render domain reset. - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.
	0	**Initiate Graphics Full Soft Reset** Access: R/W Set Graphics Full Soft-Reset Control: '1' : Initiate a full graphics reset (i.e., graphics render, media, and blitter reset). - Cleared by CP once the reset is complete '0' : N/A - Once set, clearing of this bit has no effect on CP. Only CP can reset this bit. Note: This is a non-posted register.

Graphics Mode Register

GFX_MODE - Graphics Mode Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000 [CHV:B, CHV:C, CHV:K]
	0x00000800 [CHV:A]
Size (in bits):	32
Trusted Type:	1
Address:	0229Ch
Valid Projects:	CHV, BSW
Address:	1229Ch-1229Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VCSUNIT0
Address:	1A29Ch-1A29Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VECSUNIT
Address:	1C29Ch-1C29Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_VCSUNIT1
Address:	2229Ch-2229Fh
Name:	Graphics Mode Register
ShortName:	GFX_MODE_BCSUNIT

Description
This register contains a control bit for the new execlist and 2-level PPGTT functions.
DefaultValue = 00002800h

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**Execlist Enable** Mask: MMIO#31 When set, software can utilize the execlist registers to load a context into hardware. MI_SET_CONTEXT and MI_ARB_CHECK commands will be converted to NOOP if parsed. When this bit is clear, the Execlist mechanism cannot be used. The context must be loaded via MI_SET_CONTEXT and the ring must be loaded via MMIO access.

GFX_MODE - Graphics Mode Register

		Programming Notes This bit is *not* intended to be changed dynamically. Changing the value of this bit while rendering is in progress will have UNDEFINED results. This bit should be changed only after a full reset and before submitting *any* commands to the device.
	14	**Reserved** Project: CHV, BSW
	13	**Flush TLB invalidation Mode** Project: CHV, BSW Format: U1 This field controls the invalidation if the TLB cache inside the hardware. When enabled this bit limits the invalidation of the TLB only to batch buffer boundaries, to pipe_control commands which have the TLB invalidation bit set and sync flushes. If disabled, the TLB caches are flushed for every full flush of the pipeline.
	12	**Reserved** Project: All Format: MBZ
	11	**Reserved** Project: CHV, BSW Format: MBZ
	11	**Replay Mode** Project: CHV, BSW Format: U1 Context Switch Granularity This field controls the granularity of the replay mechanism when coming back into a previously preempted context.

Value	Name	Description
1h	Object Level Preemption **[Default]**	Object Level. Preemption is done on an Object Level Boundary in VF. Objects send down by VF are completely rendered. Pipeline is flushed before switching to the next context. On resubmission of the context VF starts parsing form the object where it got preempted last time.
0h	mid-cmdbuffer preemption	Drawcall Level. Pipeline is flushed before switching to the next context. Commands parsed are committed to completing before a context switch.

Programming Notes
A fixed function pipe flush is required before modifying this field.
The replay mode must be set to 0.

	10	**Reserved** Project: All

GFX_MODE - Graphics Mode Register

Format:	MBZ

9 **Per-Process GTT Enable**

Project:	CHV, BSW
Format:	Enabled

Per-Process GTT Enable

Value	Name	Description
0h	PPGTT Disable **[Default]**	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.

Programming Notes

This bit is used for enabling PPGTT access in Ring Buffer mode of scheduling. Privilege field in context descriptor states the same in Execlist Mode of scheduling. This field should be set before programming PDP0/1/2/3 registers in order to set the PPGTT translation of memory access.
Programming this bit doesn't enable or disable the PPGTT translation of memory access immediately; the change comes in to affect only when the Page Directory registers are programmed. Programming this bit must be followed by programming Page Directory Registers in order to enable or disable the PPGTT translation of memory access.

8 **Reserved**

Project:	CHV, BSW

7 **64Bit Virtual Addressing Enable**

Project:	CHV, BSW
Format:	Enabled

Per-Process GTT Enable

Value	Name	Description
0h	64Bit Virtual Addressing Disable **[Default]**	When clear indicates GFX operating in 32bit Virtual Addressing for PPGTT based memory access.

Programming Notes

This bit is only valid when PPGTT is enabled in ring buffer mode of scheduling. Context Descriptor has a similar bit to control 64bit virtual addressing in execlist mode of scheduling. Whether this field is set or clear virtual addresses translated through GGTT are always 32Bit. This field should be programmed before enabling PPGTT access. When this field is not set or for GGTT virtual addresses, Graphics Address [47:32] field of any commands or register exercised by SW should be programmed to 0x0.

64 Bit Virtual Addressing is not supported on CHV, BSW and must not be set.

GFX_MODE - Graphics Mode Register

	6:5	**Reserved** Project: CHV, BSW
	4	**Reserved** Project: CHV, BSW
	3:1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Privilege Check Disable** Project: CHV, BSW Format: Enable This field when set, disables Privilege Violation checks on non-privileged batch buffers. When set Privileged commands are allowed to be executed from non-privileged batch buffers. **Workaround** Workaround: Irrespective of "Privilege Check Disable" bit set, HW enforces chained or second level batch buffer "Address Space Indicator" to be PPGTT if the parent batch buffer Address Space Indicator is PPGTT.

GS Domain Clock Gate Control Register

GSCKGCTL - GS Domain Clock Gate Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09028h

Gunit Graphics Slow Clock Domain Clock Gating Control Register

DWord	Bit	Description
0	31:29	**RSVD** Access: RO RSVD
	28:24	**GMBC Clock Gate IDLE Count** Access: R/W GMBC IDLE Timer Count Value - bits[27:24] 0 = GMBC Idle timer disabled 1-15 = After GMBC indicates idle, wait this many clocks before gating. Bit [28] is additional Reserved R/W
	23:20	**GPMG Clock Gate IDLE Counter** Access: R/W GPMGi dle timer. GPMG uses this as a its idle counter pre-load value for clock gating 0 = Idle timer disabled 1 - 15 = After unit indicates its idle, wait this many clocks before gating
	19:16	**GSIDLECNT** Access: R/W Generic GUNIT GSCLK domain idle timer. GUNIT GS fubs (other than GMBC) use this as a counter pre-load value 0 = Idle timer disabled 1 - 15 = After a unit indicates idle, wait this many clocks before gating.
	15:13	**RSVD** Access: RO Reserved
	12	**GDTCKGEN** Access: R/W

GSCKGCTL - GS Domain Clock Gate Control Register

		GDT Clock Gating Enable. 0 = Disable clock gating. 1 = Enable clock gating.
	11	**GMBCPCKDIS** Access: R/W GMBC Performance Monitor Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	10	**GMBCCKGDIS** Access: R/W GMBC Clock Gating Disable: controls non-performance monitoring related clocks in the GMBC 0 = Enable clock gating 1 = Disable clock gating
	9	**GPMGCKDIS** Access: R/W GPMG Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	8	**G3DRDCKGDIS** Access: R/W G3D Read Return Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	7	**G3DOUTCKGDIS** Access: R/W G3D Outbound Clock Gating Disable: controls clocks associated with outbound requests. 0 = Enable clock gating 1 = Disable clock gating
	6	**GMCFGCKGDIS** Access: R/W GMCFG Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	5	**GKEYSCKGDIS** Access: R/W

GSCKGCTL - GS Domain Clock Gate Control Register

		GKEYS Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	4	**GCCBGSGDIS** Access: R/W GCCBGS Clock Gating Disable 0 = Enable clock gating 1 = Disable clock gating
	3:0	**RSVD** Access: RO Reserved

GS Invocation Counter

GS_INVOCATION_COUNT - GS Invocation Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02328h

This register stores the number of objects that are part of geometry shader threads. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**GS Invocation Count UDW** Number of objects that are dispatched as a geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)
	31:0	**GS Invocation Count LDW** Number of objects that are dispatched as a geometry shader threads invoked by the GS stage. Updated only when **Statistics Enable** is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)

GS Primitives Counter

GS_PRIMITIVES_COUNT - GS Primitives Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02330h

This register reflects the total number of primitives that have been output by the Geometry Shader stage. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**GS Primitives Count UDW** Total number of primitives output by the geometry stage. Updated only when Statistics Enable is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)
	31:0	**GS Primitives Count LDW** Total number of primitives output by the geometry stage. Updated only when Statistics Enable is set in GS_STATE (see the Geometry Shader Chapter in the 3D Volume.)

GT_CR_POWER_METER_CTRL

PWRMTRLK - GT_CR_POWER_METER_CTRL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A248h-0A24Bh

DWord	Bit	Description
0	31	**Power Meter and Push Bus Control Lock** Access: R/W Lock Controls whether power meter weights and other push bus control registers are writeable. 0 : Power Meter weights and control registers are writeable. 1: Writes to Power Meter weights and control registers are blocked. Lock bit cannot be cleared without cold reset.
	30:7	**Reserved** Access: RO
	6	**GSclk Domain GTI Baseline Energy Count Enable** Access: R/W Lock Controls whether the energy counter for the GTI power well, counting in the GSclk domain, is running or not. 0: Accumulator in PM is disabled 1: Accumulator in PM is enabled
	5	**Reserved** Access: RO
	4	**Render Power Meter Counter Enable** Access: R/W Lock Render Power Meter Enable : Controls whether power meter is running or not. 0: All Power meter counters/accumulators in PM are disabled and held at zero. 1: All power meter counters/accumulators in PM are enabled. Note that this bit does not affect intermediate accumulation/overflow logic elsewhere in the Gfx engine.
	3	**Media Power Meter Counter Enable** Access: R/W Lock Media Power Meter Enable :

PWRMTRLK - GT_CR_POWER_METER_CTRL			
		Controls whether power meter is running or not. 0: All Power meter counters/accumulators in PM are disabled and held at zero. 1: All power meter counters/accumulators in PM are enabled. Note that this bit does not affect intermediate accumulation/overflow logic elsewhere in the Gfx engine.	
	2:0	**Reserved**	
		Access:	RO

GTFIFOCTL

GTFIFOCTL - GTFIFOCTL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	120008h

GT FIFO Control.

DWord	Bit	Description
0	31	**GT_IOSFSB_READ_POLICY** Default Value: 0b Access: R/W The SEC must first write this register bit before attempting Gunit register context reads. This is identified usage model requiring IOSF SB reads. Note : For CHV, BSW, Punit has it's own IOSF SB P interface. 0 (Default) : Abort IOSF SB reads. Only Kf1 and Kf1 status are allowed. 1 : Allow IOSF SB reads.
	30:16	**SPARE15** Default Value: 0000h Access: RO Reserved
	15	**GT_FIFO_SB_POLICY** Default Value: 0b Access: R/W GT_WakeFIFO IOSF SB Policy register 0 (default) : If WakeFIFO threshold hit and dedicated IOSF SB buffering is full, stall IOSF SB accesses targeting WakeFIFO. 1 : If WakeFIFO threshold is hit AND dedicated IOSF SB buffering is full : a. Drop IOSF SB write requests to WakeFIFO b. IOSF SB reads targeting WakeFIFO return 1s. Evaluated for the request at the head of the IA_WakeFIFO.
	14	**GT_FIFO_BLOB_POLICY** Default Value: 0b Access: R/W
	13	**GT_FIFO_PRI_POLICY** Default Value: 0b Access: R/W GT_WakeFIFO IOSF Primary Policy register. 0 (Default) : If WakeFIFO threshold hit, stall IOSF Primary accesses targeting WakeFIFO.

GTFIFOCTL - GTFIFOCTL

		1 : If WakeFIFO threshold is hit : a. Drop IOSF Primary writes to WakeFIFO b. IOSF Primary reads targeting WakeFIFO return 1s. Don't hang the system, but device 2 may hang.
	12	**Block all Policy** Default Value: 0b Access: R/W BlockALL policy register applies to both HW FIFO and IA GT FIFO. This bit only applies when the GPM BlockALL indication is asserted. GPM BlockALL indication can assert for : A) CPD and B) for a period of time during RC6 entry. 0 (default) : Allow register collapsing and stall request at the head if it is not collapse-able. 1 : Stall request at the head.
	11	**RC6_POLICY** Default Value: 0b Access: R/W RC6 policy register applies to both HW FIFO and IA GT FIFO. This bit only applies : - when BlockALL is not asserted - within RC6, and - ONLY when AllowWake register (13_0090h[0]=1). 0 (default) : Allow register collapsing and drop request at the head if it is not collapsible. 1 : Stall request at the head if within RC6. Note: All hardware initiated requests should be collapsible. Note: Starting with Gen8 Gfx, the driver is required to ensure the targeted power well is alive before initiating an access outside shadow register space.
	10:9	**SPARE2** Default Value: 00b Access: R/W spare bit
	8	**Reserved** Default Value: 0b Access: R/W
	7:0	**SPARE8** Default Value: 00h Access: RO Reserved

GT Function Level Reset Control Message

FLRCTLMSG - GT Function Level Reset Control Message

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	08100h

GT FLR Control Register

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:1	**Reserved** Access: RO Reserved
	0	**Reserved** Access: RO Reserved R/W: FLR is not supported on CHV, BSW.

GT INTERRUPT 0 ENABLE REGISTER

GT_INTERRUPT0_IER - GT INTERRUPT 0 ENABLE REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4430Ch-4430Fh

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for Render CS. Bits 31:16 are used for Blitter CS.
The IER enabled Render Interrupt IIR (sticky) bits are ORed together to generate the Render Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled Blitter Interrupt IIR (sticky) bits are ORed together to generate the Blitter Interrupts Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W
	30	**UNUSED1** Access: R/W
	29	**UNUSED2** Access: R/W
	28	**UNUSED3** Access: R/W
	27	**BCS_WAIT_ON_SEMAPHORE** Access: R/W BCS wait on semaphore
	26	**UNUSED4** Access: R/W
	25	**UNUSED5** Access: R/W
	24	**BCS_CTX_SWITCH_INTERRUPT** Access: R/W BCS context switch interrupt
	23	**UNUSED6** Access: R/W

GT_INTERRUPT0_IER - GT INTERRUPT 0 ENABLE REGISTER

22	**UNUSED7**	
	Access:	R/W
21	**UNUSED8**	
	Access:	R/W
20	**BCS_MI_FLUSH_DWNOTIFY**	
	Access:	R/W
	BCS MI flush DW notify	
19	**BCS_ERROR_INTERRUPT**	
	Access:	R/W
	BCS error interrupt	
18	**UNUSED9**	
	Access:	R/W
17	**UNUSED10**	
	Access:	R/W
16	**BCS_MI_USER_INTERRUPT**	
	Access:	R/W
	BCS MI user interrupt	
15	**UNUSED11**	
	Access:	R/W
14	**UNUSED12**	
	Access:	R/W
13	**UNUSED13**	
	Access:	R/W
12	**UNUSED14**	
	Access:	R/W
11	**CS_WAIT_ON_SEMAPHORE**	
	Access:	R/W
	CS wait on semaphore	
10	**CS_L3_COUNTER_SAVE**	
	Access:	R/W
	CS L3 counter save	
9	**UNUSED15**	
	Access:	R/W

GT_INTERRUPT0_IER - GT INTERRUPT 0 ENABLE REGISTER

	8	**CS_CTX_SWITCH_INTERRUPT** Access: R/W CS context switch interrupt
	7	**PAGE_FAULT_ERROR** Access: R/W this interrupt is for handling legacy page fault interface for all command streamer (BCS, VCS, RCS, VECS). when fault repair mode is enabled, interrupt mask register value is not looked at to generate interrupt due to page fault. please refer to 'page fault support' section for more details.
	6	**CS_WATCHDOG_COUNTER_EXPIRED** Access: R/W CS watchdog counter expired
	5	**L3PARITYERROR** Access: R/W L3 parity error
	4	**CS_PIPE_CONTROL_NOTIFY** Access: R/W CS pipe control notify
	3	**CS_ERROR_INTERRUPT** Access: R/W CS error interrupt
	2	**UNUSED17** Access: R/W
	1	**Reserved**
	0	**CS_MI_USER_INTERRUPT** Access: R/W CS context switch interrupt

GT INTERRUPT 0 IDENTITY REGISTER

GT_INTERRUPT0_IIR - GT INTERRUPT 0 IDENTITY REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44308h-4430Bh

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for Render CS. Bits 31:16 are used for Blitter CS.
The IER enabled Render Interrupt IIR (sticky) bits are ORed together to generate the Render Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled Blitter Interrupt IIR (sticky) bits are ORed together to generate the Blitter Interrupts Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W One Clear
	30	**UNUSED1** Access: R/W One Clear
	29	**UNUSED2** Access: R/W One Clear
	28	**UNUSED3** Access: R/W One Clear
	27	**BCS_WAIT_ON_SEMAPHORE** Access: R/W One Clear BCS wait on semaphore
	26	**UNUSED4** Access: R/W One Clear
	25	**UNUSED5** Access: R/W One Clear
	24	**BCS_CTX_SWITCH_INTERRUPT** Access: R/W One Clear BCS context switch interrupt
	23	**UNUSED6** Access: R/W One Clear

GT_INTERRUPT0_IIR - GT INTERRUPT 0 IDENTITY REGISTER

22	**UNUSED7**	
	Access:	R/W One Clear
21	**UNUSED8**	
	Access:	R/W One Clear
20	**BCS_MI_FLUSH_DWNOTIFY**	
	Access:	R/W One Clear
	BCS MI flush DW notify	
19	**BCS_ERROR_INTERRUPT**	
	Access:	R/W One Clear
	BCS error interrupt	
18	**UNUSED9**	
	Access:	R/W One Clear
17	**UNUSED10**	
	Access:	R/W One Clear
16	**BCS_MI_USER_INTERRUPT**	
	Access:	R/W One Clear
	BCS MI user interrupt	
15	**UNUSED11**	
	Access:	R/W One Clear
14	**UNUSED12**	
	Access:	R/W One Clear
13	**UNUSED13**	
	Access:	R/W One Clear
12	**UNUSED14**	
	Access:	R/W One Clear
11	**CS_WAIT_ON_SEMAPHORE**	
	Access:	R/W One Clear
	CS wait on semaphore	
10	**CS_L3_COUNTER_SAVE**	
	Access:	R/W One Clear
	CS L3 counter save	
9	**UNUSED15**	
	Access:	R/W One Clear

GT_INTERRUPT0_IIR - GT INTERRUPT 0 IDENTITY REGISTER

	8	**CS_CTX_SWITCH_INTERRUPT** Access: R/W One Clear CS context switch interrupt
	7	**PAGE_FAULT_ERROR** Access: R/W One Clear this interrupt is for handling legacy page fault interface for all command streamer (BCS, VCS, RCS, VECS). when fault repair mode is enabled, interrupt mask register value is not looked at to generate interrupt due to page fault. please refer to 'page fault support' section for more details.
	6	**CS_WATCHDOG_COUNTER_EXPIRED** Access: R/W One Clear CS watchdog counter expired
	5	**L3PARITYERROR** Access: R/W One Clear L3 parity error
	4	**CS_PIPE_CONTROL_NOTIFY** Access: R/W One Clear CS pipe control notify
	3	**CS_ERROR_INTERRUPT** Access: R/W One Clear CS error interrupt
	2	**UNUSED17** Access: R/W One Clear
	1	**Reserved**
	0	**CS_MI_USER_INTERRUPT** Access: R/W One Clear CS context switch interrupt

GT INTERRUPT 0 MASK REGISTER

GT_INTERRUPT0_IMR - GT INTERRUPT 0 MASK REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x09190DF9
Size (in bits):	32
Address:	44304h-44307h

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for Render CS. Bits 31:16 are used for Blitter CS.
The IER enabled Render Interrupt IIR (sticky) bits are ORed together to generate the Render Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled Blitter Interrupt IIR (sticky) bits are ORed together to generate the Blitter Interrupts Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W
	30	**UNUSED1** Access: R/W
	29	**UNUSED2** Access: R/W
	28	**UNUSED3** Access: R/W
	27	**BCS_WAIT_ON_SEMAPHORE** Default Value: 1b Access: R/W BCS wait on semaphore
	26	**UNUSED4** Access: R/W
	25	**UNUSED5** Access: R/W
	24	**BCS_CTX_SWITCH_INTERRUPT** Default Value: 1b Access: R/W BCS context switch interrupt
	23	**UNUSED6**

GT_INTERRUPT0_IMR - GT INTERRUPT 0 MASK REGISTER

	Access:	R/W
22	**UNUSED7**	
	Access:	R/W
21	**UNUSED8**	
	Access:	R/W
20	**BCS_MI_FLUSH_DWNOTIFY**	
	Default Value:	1b
	Access:	R/W
	BCS MI flush DW notify	
19	**BCS_ERROR_INTERRUPT**	
	Default Value:	1b
	Access:	R/W
	BCS error interrupt	
18	**UNUSED9**	
	Access:	R/W
17	**UNUSED10**	
	Access:	R/W
16	**BCS_MI_USER_INTERRUPT**	
	Default Value:	1b
	Access:	R/W
	BCS MI user interrupt	
15	**UNUSED11**	
	Access:	R/W
14	**UNUSED12**	
	Access:	R/W
13	**UNUSED13**	
	Access:	R/W
12	**UNUSED14**	
	Access:	R/W
11	**CS_WAIT_ON_SEMAPHORE**	
	Default Value:	1b
	Access:	R/W
	CS wait on semaphore	

GT_INTERRUPT0_IMR - GT INTERRUPT 0 MASK REGISTER

10	**CS_L3_COUNTER_SAVE**	
	Default Value:	1b
	Access:	R/W
	CS L3 counter save	
9	**UNUSED15**	
	Access:	R/W
8	**CS_CTX_SWITCH_INTERRUPT**	
	Default Value:	1b
	Access:	R/W
	CS context switch interrupt	
7	**PAGE_FAULT_ERROR**	
	Default Value:	1b
	Access:	R/W
	this interrupt is for handling legacy page fault interface for all command streamer (BCS, VCS, RCS, VECS). when fault repair mode is enabled, interrupt mask register value is not looked at to generate interrupt due to page fault. please refer to 'page fault support' section for more details.	
6	**CS_WATCHDOG_COUNTER_EXPIRED**	
	Default Value:	1b
	Access:	R/W
	CS watchdog counter expired	
5	**L3PARITYERROR**	
	Default Value:	1b
	Access:	R/W
	L3 parity error	
4	**CS_PIPE_CONTROL_NOTIFY**	
	Default Value:	1b
	Access:	R/W
	CS pipe control notify	
3	**CS_ERROR_INTERRUPT**	
	Default Value:	1b
	Access:	R/W
	CS error interrupt	

GT_INTERRUPT0_IMR - GT INTERRUPT 0 MASK REGISTER

	2	**UNUSED17** Access: R/W
	1	**Reserved**
	0	**CS_MI_USER_INTERRUPT** Default Value: 1b Access: R/W CS context switch interrupt

GT INTERRUPT 0 STATUS REGISTER

GT_INTERRUPT0_ISR - GT INTERRUPT 0 STATUS REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44300h-44303h

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for Render CS. Bits 31:16 are used for Blitter CS.
The IER enabled Render Interrupt IIR (sticky) bits are ORed together to generate the Render Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled Blitter Interrupt IIR (sticky) bits are ORed together to generate the Blitter Interrupts Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: RO
	30	**UNUSED1** Access: RO
	29	**UNUSED2** Access: RO
	28	**UNUSED3** Access: RO
	27	**BCS_WAIT_ON_SEMAPHORE** Access: RO BCS wait on semaphore
	26	**UNUSED4** Access: RO
	25	**UNUSED5** Access: RO
	24	**BCS_CTX_SWITCH_INTERRUPT** Access: RO BCS context switch interrupt
	23	**UNUSED6** Access: RO

GT_INTERRUPT0_ISR - GT INTERRUPT 0 STATUS REGISTER

22	**UNUSED7**	
	Access:	RO
21	**UNUSED8**	
	Access:	RO
20	**BCS_MI_FLUSH_DWNOTIFY**	
	Access:	RO
	BCS MI flush DW notify	
19	**BCS_ERROR_INTERRUPT**	
	Access:	RO
	BCS error interrupt	
18	**UNUSED9**	
	Access:	RO
17	**UNUSED10**	
	Access:	RO
16	**BCS_MI_USER_INTERRUPT**	
	Access:	RO
	BCS MI user interrupt	
15	**UNUSED11**	
	Access:	RO
14	**UNUSED12**	
	Access:	RO
13	**UNUSED13**	
	Access:	RO
12	**UNUSED14**	
	Access:	RO
11	**CS_WAIT_ON_SEMAPHORE**	
	Access:	RO
	CS wait on semaphore	
10	**CS_L3_COUNTER_SAVE**	
	Access:	RO
	CS L3 counter save	
9	**UNUSED15**	
	Access:	RO

GT_INTERRUPT0_ISR - GT INTERRUPT 0 STATUS REGISTER

	8	**CS_CTX_SWITCH_INTERRUPT** Access: RO CS context switch interrupt
	7	**PAGE_FAULT_ERROR** Access: RO this interrupt is for handling legacy page fault interface for all command streamer (BCS, VCS, RCS, VECS). when fault repair mode is enabled, interrupt mask register value is not looked at to generate interrupt due to page fault. please refer to 'page fault support' section for more details.
	6	**CS_WATCHDOG_COUNTER_EXPIRED** Access: RO CS watchdog counter expired
	5	**L3PARITYERROR** Access: RO L3 parity error
	4	**CS_PIPE_CONTROL_NOTIFY** Access: RO CS pipe control notify
	3	**CS_ERROR_INTERRUPT** Access: RO CS error interrupt
	2	**UNUSED17** Access: RO
	1	**Reserved**
	0	**CS_MI_USER_INTERRUPT** Access: RO CS context switch interrupt

GT INTERRUPT1 ENABLE REGISTER

GT_INTERRUPT1_IER - GT INTERRUPT1 ENABLE REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4431Ch-4431Fh

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for VCS1. Bits 31:16 are used for VCS2.
The IER enabled VCS1 Interrupt IIR (sticky) bits are ORed together to generate the VCS1 Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled VCS2 Interrupt IIR (sticky) bits are ORed together to generate the VCS2 Interrupt Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W
	30	**UNUSED1** Access: R/W
	29	**UNUSED2** Access: R/W
	28	**UNUSED3** Access: R/W
	27	**VCS2_WAIT_ON_SEMAPHORE** Access: R/W VCS2 wait on semaphore
	26	**UNUSED4** Access: R/W
	25	**Reserved**
	24	**VCS2_CTX_SWITCH_INTERRUPT** Access: R/W VCS2 context switch interrupt
	23	**UNUSED5** Access: R/W
	22	**VCS2_WATCHDOG_COUNTER_EXPIRED**

GT_INTERRUPT1_IER - GT INTERRUPT1 ENABLE REGISTER

		Access: R/W VCS2 watchdog counter expired
	21	**Reserved**
	20	**VCS2_MI_FLUSH_DWNOTIFY** Access: R/W VCS2 MI flush DW notify
	19	**VCS2_ERROR_INTERRUPT** Access: R/W VCS2 error interrupt
	18	**UNUSED6** Access: R/W
	17	**UNUSED7** Access: R/W
	16	**VCS2_MI_USER_INTERRUPT** Access: R/W VCS2 MI user interrupt
	15	**UNUSED8** Access: R/W
	14	**UNUSED9** Access: R/W
	13	**UNUSED10** Access: R/W
	12	**UNUSED11** Access: R/W
	11	**VCS1_WAIT_ON_SEMAPHORE** Access: R/W VCS1 wait on semaphore
	10	**UNUSED12** Access: R/W
	9	**Reserved**
	8	**VCS1_CTX_SWITCH_INTERRUPT** Access: R/W VCS1 context switch interrupt

GT_INTERRUPT1_IER - GT INTERRUPT1 ENABLE REGISTER

	7	**UNUSED13** Access: R/W
	6	**VCS1_WATCHDOG_COUNTER_EXPIRED** Access: R/W VCS1 watchdog counter expired
	5	**Reserved**
	4	**VCS1_MI_FLUSH_DWNOTIFY** Access: R/W VCS1 MI flush DW notify
	3	**VCS1_ERROR_INTERRUPT** Access: R/W VCS1 error interrupt
	2	**UNUSED14** Access: R/W
	1	**UNUSED15** Access: R/W
	0	**VCS1_MI_USER_INTERRUPT** Access: R/W VCS1 MI user interrupt

GT INTERRUPT1 IDENTITY REGISTER

GT_INTERRUPT1_IIR - GT INTERRUPT1 IDENTITY REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44318h-4431Bh

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for VCS1. Bits 31:16 are used for VCS2.
The IER enabled VCS1 Interrupt IIR (sticky) bits are ORed together to generate the VCS1 Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled VCS2 Interrupt IIR (sticky) bits are ORed together to generate the VCS2 Interrupt Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W One Clear
	30	**UNUSED1** Access: R/W One Clear
	29	**UNUSED2** Access: R/W One Clear
	28	**UNUSED3** Access: R/W One Clear
	27	**VCS2_WAIT_ON_SEMAPHORE** Access: R/W One Clear VCS2 wait on semaphore
	26	**UNUSED4** Access: R/W One Clear
	25	**Reserved**
	24	**VCS2_CTX_SWITCH_INTERRUPT** Access: R/W One Clear VCS2 context switch interrupt
	23	**UNUSED5** Access: R/W One Clear
	22	**VCS2_WATCHDOG_COUNTER_EXPIRED**

GT_INTERRUPT1_IIR - GT INTERRUPT1 IDENTITY REGISTER

		Access: R/W One Clear VCS2 watchdog counter expired
	21	**Reserved**
	20	**VCS2_MI_FLUSH_DWNOTIFY** Access: R/W One Clear VCS2 MI flush DW notify
	19	**VCS2_ERROR_INTERRUPT** Access: R/W One Clear VCS2 error interrupt
	18	**UNUSED6** Access: R/W One Clear
	17	**UNUSED7** Access: R/W One Clear
	16	**VCS2_MI_USER_INTERRUPT** Access: R/W One Clear VCS2 MI user interrupt
	15	**UNUSED8** Access: R/W One Clear
	14	**UNUSED9** Access: R/W One Clear
	13	**UNUSED10** Access: R/W One Clear
	12	**UNUSED11** Access: R/W One Clear
	11	**VCS1_WAIT_ON_SEMAPHORE** Access: R/W One Clear VCS1 wait on semaphore
	10	**UNUSED12** Access: R/W One Clear
	9	**Reserved**
	8	**VCS1_CTX_SWITCH_INTERRUPT** Access: R/W One Clear VCS1 context switch interrupt

GT_INTERRUPT1_IIR - GT INTERRUPT1 IDENTITY REGISTER

	7	**UNUSED13** Access: R/W One Clear
	6	**VCS1_WATCHDOG_COUNTER_EXPIRED** Access: R/W One Clear VCS1 watchdog counter expired
	5	**Reserved**
	4	**VCS1_MI_FLUSH_DWNOTIFY** Access: R/W One Clear VCS1 MI flush DW notify
	3	**VCS1_ERROR_INTERRUPT** Access: R/W One Clear VCS1 error interrupt
	2	**UNUSED14** Access: R/W One Clear
	1	**UNUSED15** Access: R/W One Clear
	0	**VCS1_MI_USER_INTERRUPT** Access: R/W One Clear VCS1 MI user interrupt

GT INTERRUPT1 MASK REGISTER

GT_INTERRUPT1_IMR - GT INTERRUPT1 MASK REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x09590959
Size (in bits):	32
Address:	44314h-44317h

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for VCS1. Bits 31:16 are used for VCS2.
The IER enabled VCS1 Interrupt IIR (sticky) bits are ORed together to generate the VCS1 Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled VCS2 Interrupt IIR (sticky) bits are ORed together to generate the VCS2 Interrupt Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: R/W
	30	**UNUSED1** Access: R/W
	29	**UNUSED2** Access: R/W
	28	**UNUSED3** Access: R/W
	27	**VCS2_WAIT_ON_SEMAPHORE** Default Value: 1b Access: R/W VCS2 wait on semaphore
	26	**UNUSED4** Access: R/W
	25	**Reserved**
	24	**VCS2_CTX_SWITCH_INTERRUPT** Default Value: 1b Access: R/W VCS2 context switch interrupt
	23	**UNUSED5** Access: R/W

GT_INTERRUPT1_IMR - GT INTERRUPT1 MASK REGISTER

	22	**VCS2_WATCHDOG_COUNTER_EXPIRED** Default Value: 1b Access: R/W VCS2 watchdog counter expired
	21	**Reserved**
	20	**VCS2_MI_FLUSH_DWNOTIFY** Default Value: 1b Access: R/W VCS2 MI flush DW notify
	19	**VCS2_ERROR_INTERRUPT** Default Value: 1b Access: R/W VCS2 error interrupt
	18	**UNUSED6** Access: R/W
	17	**UNUSED7** Access: R/W
	16	**VCS2_MI_USER_INTERRUPT** Default Value: 1b Access: R/W VCS2 MI user interrupt
	15	**UNUSED8** Access: R/W
	14	**UNUSED9** Access: R/W
	13	**UNUSED10** Access: R/W
	12	**UNUSED11** Access: R/W
	11	**VCS1_WAIT_ON_SEMAPHORE** Default Value: 1b Access: R/W VCS1 wait on semaphore

GT_INTERRUPT1_IMR - GT INTERRUPT1 MASK REGISTER

	10	**UNUSED12** Access: R/W
	9	**Reserved**
	8	**VCS1_CTX_SWITCH_INTERRUPT** Default Value: 1b Access: R/W VCS1 context switch interrupt
	7	**UNUSED13** Access: R/W
	6	**VCS1_WATCHDOG_COUNTER_EXPIRED** Default Value: 1b Access: R/W VCS1 watchdog counter expired
	5	**Reserved**
	4	**VCS1_MI_FLUSH_DWNOTIFY** Default Value: 1b Access: R/W VCS1 MI flush DW notify
	3	**VCS1_ERROR_INTERRUPT** Default Value: 1b Access: R/W VCS1 error interrupt
	2	**UNUSED14** Access: R/W
	1	**UNUSED15** Access: R/W
	0	**VCS1_MI_USER_INTERRUPT** Default Value: 1b Access: R/W VCS1 MI user interrupt

GT INTERRUPT1 STATUS REGISTER

GT_INTERRUPT1_ISR - GT INTERRUPT1 STATUS REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44310h-44313h

This table indicates which events are mapped to each bit of the GT Interrupt 0 registers. Bits 15:0 are used for VCS1. Bits 31:16 are used for VCS2.
The IER enabled VCS1 Interrupt IIR (sticky) bits are ORed together to generate the VCS1 Interrupts Pending bit in the Master Interrupt Control register.
The IER enabled VCS2 Interrupt IIR (sticky) bits are ORed together to generate the VCS2 Interrupt Pending bit in the Master Interrupt Control register.

DWord	Bit	Description
0	31	**UNUSED0** Access: RO
	30	**UNUSED1** Access: RO
	29	**UNUSED2** Access: RO
	28	**UNUSED3** Access: RO
	27	**VCS2_WAIT_ON_SEMAPHORE** Access: RO VCS2 wait on semaphore
	26	**UNUSED4** Access: RO
	25	**Reserved**
	24	**VCS2_CTX_SWITCH_INTERRUPT** Access: RO VCS2 context switch interrupt
	23	**UNUSED5** Access: RO
	22	**VCS2_WATCHDOG_COUNTER_EXPIRED**

GT_INTERRUPT1_ISR - GT INTERRUPT1 STATUS REGISTER

		Access: RO VCS2 watchdog counter expired
	21	**Reserved**
	20	**VCS2_MI_FLUSH_DWNOTIFY** Access: RO VCS2 MI flush DW notify
	19	**VCS2_ERROR_INTERRUPT** Access: RO VCS2 error interrupt
	18	**UNUSED6** Access: RO
	17	**UNUSED7** Access: RO
	16	**VCS2_MI_USER_INTERRUPT** Access: RO VCS2 MI user interrupt
	15	**UNUSED8** Access: RO
	14	**UNUSED9** Access: RO
	13	**UNUSED10** Access: RO
	12	**UNUSED11** Access: RO
	11	**VCS1_WAIT_ON_SEMAPHORE** Access: RO VCS1 wait on semaphore
	10	**UNUSED12** Access: RO
	9	**Reserved**
	8	**VCS1_CTX_SWITCH_INTERRUPT** Access: RO VCS1 context switch interrupt

GT_INTERRUPT1_ISR - GT INTERRUPT1 STATUS REGISTER

	7	**UNUSED13** Access: RO
	6	**VCS1_WATCHDOG_COUNTER_EXPIRED** Access: RO VCS1 watchdog counter expired
	5	**Reserved**
	4	**VCS1_MI_FLUSH_DWNOTIFY** Access: RO VCS1 MI flush DW notify
	3	**VCS1_ERROR_INTERRUPT** Access: RO VCS1 error interrupt
	2	**UNUSED14** Access: RO
	1	**UNUSED15** Access: RO
	0	**VCS1_MI_USER_INTERRUPT** Access: RO VCS1 MI user interrupt

GT INTERRUPT3 ENABLE REGISTER

GT_INTERRUPT3_IER - GT INTERRUPT3 ENABLE REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4433Ch-4433Fh

This table indicates which events are mapped to each bit of the GT interrupt 3 register. The VEDBOX Interrupt IIR sticky bits are Or'ed together to generate VEDBOX interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:29	**UNUSED0** Access: R/W
	28	**Reserved**
	27:17	Reserved
	16	Reserved
	15:12	**UNUSED2** Access: R/W
	11	**VECS_WAIT_SEMAPHORE** Access: R/W VECS wait on semaphore
	10:9	**UNUSED3** Access: R/W
	8	**VECS_CTX_SWITCH_INT** Access: R/W VECS context switch interrupt
	7:5	**UNUSED4** Access: R/W
	4	**VECS_MI_FLUSH_DWNOTIFY** Access: R/W VECS MI Flush DW Notify
	3	**VECS_ERR_INT** Access: R/W VECS error interrupt

GT_INTERRUPT3_IER - GT INTERRUPT3 ENABLE REGISTER		
	2:1	**UNUSED5** Access: R/W
	0	**VECS_MI_USER_INT** Access: R/W VECS MI user interrupt

GT INTERRUPT3 IDENTITY REGISTER

GT_INTERRUPT3_IIR - GT INTERRUPT3 IDENTITY REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44338h-4433Bh

This table indicates which events are mapped to each bit of the GT interrupt 3 register. The VEDBOX Interrupt IIR sticky bits are Ored together to generate VEDBOX interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:29	**UNUSED0** Access: R/W One Clear
	28	**Reserved**
	27:17	Reserved
	16	Reserved
	15:12	**UNUSED2** Access: R/W One Clear
	11	**VECS_WAIT_SEMAPHORE** Access: R/W One Clear VECS wait on semaphore
	10:9	**UNUSED3** Access: R/W One Clear
	8	**VECS_CTX_SWITCH_INT** Access: R/W One Clear VECS context switch interrupt
	7:5	**UNUSED4** Access: R/W One Clear
	4	**VECS_MI_FLUSH_DWNOTIFY** Access: R/W One Clear VECS MI Flush DW Notify
	3	**VECS_ERR_INT** Access: R/W One Clear VECS error interrupt

GT_INTERRUPT3_IIR - GT INTERRUPT3 IDENTITY REGISTER

	2:1	**UNUSED5** Access: R/W One Clear
	0	**VECS_MI_USER_INT** Access: R/W One Clear VECS MI user interrupt

GT INTERRUPT3 MASK REGISTER

GT_INTERRUPT3_IMR - GT INTERRUPT3 MASK REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00010919
Size (in bits):	32
Address:	44334h-44337h

This table indicates which events are mapped to each bit of the GT interrupt 3 register. The VEDBOX Interrupt IIR sticky bits are Ored together to generate VEDBOX interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:29	**UNUSED0** Access: R/W
	28	**Reserved**
	27:17	Reserved
	16	Reserved
	15:12	**UNUSED2** Access: R/W
	11	**VECS_WAIT_SEMAPHORE** Default Value: 1b Access: R/W VECS wait on semaphore
	10:9	**UNUSED3** Access: R/W
	8	**VECS_CTX_SWITCH_INT** Default Value: 1b Access: R/W VECS context switch interrupt
	7:5	**UNUSED4** Access: R/W
	4	**VECS_MI_FLUSH_DWNOTIFY** Default Value: 1b Access: R/W VECS MI Flush DW Notify

GT_INTERRUPT3_IMR - GT INTERRUPT3 MASK REGISTER

	3	**VECS_ERR_INT** Default Value: 1b Access: R/W VECS error interrupt
	2:1	**UNUSED5** Access: R/W
	0	**VECS_MI_USER_INT** Default Value: 1b Access: R/W VECS MI user interrupt

GT INTERRUPT3 STATUS REGISTER

GT_INTERRUPT3_ISR - GT INTERRUPT3 STATUS REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44330h-44333h

This table indicates which events are mapped to each bit of the GT interrupt 3 register. The VEDBOX Interrupt IIR sticky bits are Or'ed together to generate VEDBOX interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:29	**UNUSED0** Access: RO
	28	**Reserved**
	27:17	Reserved
	16	Reserved
	15:12	**UNUSED2** Access: RO
	11	**VECS_WAIT_SEMAPHORE** Access: RO VECS wait on semaphore
	10:9	**UNUSED3** Access: RO
	8	**VECS_CTX_SWITCH_INT** Access: RO VECS context switch interrupt
	7:5	**UNUSED4** Access: RO
	4	**VECS_MI_FLUSH_DWNOTIFY** Access: RO VECS MI Flush DW Notify
	3	**VECS_ERR_INT** Access: RO VECS error interrupt

GT_INTERRUPT3_ISR - GT INTERRUPT3 STATUS REGISTER		
	2:1	**UNUSED5** Access: RO
	0	**VECS_MI_USER_INT** Access: RO VECS MI user interrupt

GTLC_PW_STAT

GTLC_PW_STAT - GTLC_PW_STAT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130094h

This register contains Ack and status information for power well requests.

DWord	Bit	Description
0	31:8	**RESERVED** Default Value: 000000h Access: RO Reserved
	7	**Reserved**
	6	**Reserved** Default Value: 0b Access: RO Reserved
	5	**Reserved**
	4	**Reserved** Default Value: 0b Access: RO Reserved
	3:2	**Reserved** Default Value: 00b Access: RO Reserved
	1	**ALLOWWAKEERR** Default Value: 0b Access: R/W One Clear HW set, SW cleared. When access to media or render is observed when ALLOWWAKE=0, the ALLOWWAKERR bit will be set. It will be up to SW or a power cycle to clear the ALLOWWAKEERR bit.
	0	**ALLOWWAKEACK** Default Value: 0b

GTLC_PW_STAT - GTLC_PW_STAT			
		Access:	RO
		Indicates that the allow wake request has been completed.	

GTLC_SURVIVE

GTLC_SURVIVE - GTLC_SURVIVE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130098h

This register contains configuration bits for modes that the PM unit should operate in.

DWord	Bit	Description
0	31:12	**RESERVED** Default Value: 00000h Access: RO Reserved
	11	**gvd_pmu_bonus1** Default Value: 0b Access: R/W gvd_pmu_bonus1_zczfwoh
	10	**gvd_pmu_bonus0** Default Value: 0b Access: R/W gvd_pmu_bonus0_zczfwoh
	9	**pmu_gvd_bonus1** Default Value: 0b Access: RO pmu_gvd_bonus1_zczfwoh
	8	**pmu_gvd_bonus0** Default Value: 0b Access: RO pmu_gvd_bonus0_zczfwoh
	7:4	**RESERVED** Default Value: 0h Access: RO Reserved

GTLC_SURVIVE - GTLC_SURVIVE

	3	**GFXCLKSTATUS** Default Value: 0b Access: RO This bit is used as a way to confirm that the GFX clocks have been turned on with bit 2. SW would normally write a one to bit2 and then poll on this bit until.
	2	**GFXCLKFORCEON** Default Value: 0b Access: R/W When this bit is set to a '1, the gvd_pmu_gfxclockstartreq_zczfwoh signal to the Punit will be forced to a one.
	1	**SPAREBIT** Default Value: 0b Access: R/W Description: Spare bit for CHV, BSW. Project: CHV, BSW
	0	**GangMediaRender** Default Value: 0b Access: R/W 0 : The media and render power wells are brought down and up independently. 1 : PM unit will treat both wells as a unit, taking well down together and up together.

GTLC_WAKE

GTLC_WAKE - GTLC_WAKE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130090h

GT Wake Control.
This register is used as a way for the Punit to control the GTLC Render and Media Power wells.
Writing bit 0 during normal operation may result in a hang.

DWord	Bit	Description
0	31:26	**RESERVED** Default Value: 00h Access: RO Reserved
	25	**RenderContextExists** Default Value: 0b Access: RO The usage is for support of s0ix, where Punit/Driver would restore the value in this bit no later than the time a '1 is written to ALLOWWAKEREQ (bit 0 of this register) during s0ix exit. SW should not write '1 to this bit during a cold boot exit or warm reset exit. CZ reset is the only thing that can clear this bit. This bit is set by HW after the first Render context save after cold boot or warm reset. SW can only write a one to this bit. If SW tries to write a zero, the previous value will be maintained.
	24	**MediaContextExists** Default Value: 0b Access: RO The usage is for support of s0ix, where Punit/Driver would restore the value in this bit no later than the time a '1 is written to ALLOWWAKEREQ (bit 0 of this register) during s0ix exit. SW should not write '1 to this bit during a cold boot exit or warm reset exit. CZ reset is the only thing that can clear this bit. This bit is set by HW after the first Media context save after cold boot or warm reset. SW can only write a one to this bit. If SW tries to write a zero, the previous value will be maintained.
	23:8	**Common No-wake Hysteresis timer for gfxstartclkreq** Default Value: 0000h

GTLC_WAKE - GTLC_WAKE

		Access: R/W New for CHV, BSW. Hysteresis timer would be used to deassert gfxclockstartreq in case of both wells being in standby and common no-wake request occurs. Counter will be incremented on 60ns interval.
	7:1	**RESERVED** Default Value: 00h Access: RO Reserved
	0	**ALLOWWAKEREQ** Default Value: 0b Access: R/W This bit is used as a way for the driver to make sure GTLC Render AND MEDIA engines do not wake while powered down. The usage is specifically with s0ix, where the driver wants to access the message channel common well (GSclk domain) and does not want the render/media wells to wake up. To remove ambiguity, this bit should be set to '1 and the ALLOWWAKEACK should be observed to the '1 before the FWAKEMEDIAREQ/FWAKERENDERREQ are set. Care must be taken to ONLY use this register prior to Gunit being powered down for S0ix or after registers have been initialized.

GT Mode Register

GT_MODE - GT Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000 CHV, BSW
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	07008h
Valid Projects:	CHV, BSW

This Register is used to control the 6EU and 12EU configuration for GT. Writing 0x01FF01FF to this register enables the 6EU mode.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**EU Local Thread Checking Enable** Project: CHV, BSW Access: r/w This field configures the EU local thread checking. If enable the stateless access will be checked against the local thread's scratch space size and start address. Value / Name / Description / Project: 0h – Disable **[Default]** – EU local thread checking is disabled. – CHV, BSW 1h – Enable – EU local thread checking is enabled. – CHV, BSW
	14:13	**SFR mode** Project: CHV, BSW Access: r/w Format: U2 This field must be zero when not in GT4(SFR) configuration i.e GTB_rendermode fuse set to SFR.
	12	**Reserved** Project: CHV, BSW Access: r/w Format: PBC
	11	**16X16 Cross Slice Hash Disable for SF**

GT_MODE - GT Mode Register

Project:	CHV, BSW
Access:	r/w
Format:	U1

Description
This field allows to control pixel block hashing across slices.
Supports 16X16 pixel block hashing in the checker-board pattern irrespective of MSAA. Setting this bit disables hashing and therefore HW will send all the Pixels to both the slices.

Value	Name	Description
0h	Enable **[Default]**	16X16 Checkerboard hashing enabled across slices
1h	Disable	16X16 Checkerboard hashing disabled across slices

Programming Notes
Normal mode of operation in GT3 mode will be to use either 16x16 Hashing or 32x32 Hashing.

10 **16X16 Cross Slice Hash Disable**

Project:	CHV, BSW
Access:	r/w
Format:	U1

This field allows to control pixel block hashing across slices.

Value	Name	Description
0h	Enable **[Default]**	16X16 Checkerboard hashing enabled across slices
1h	Disable	16X16 Checkerboard hashing disabled across slices

9 **WIZ Hashing Mode High Bit**

Project:	CHV, BSW
Access:	r/w
Format:	U1

This field adds additional hashing modes in combination with the WIZ Hashing Mode field. The Value column in the table below refers to this field (high bit) and the WIZ Hashing Mode field (low bit). This field is don't care if the Hashing Disable bit is set.

Value	Name	Description
0h	**[Default]**	8x8 Checkerboard hashing
1h		8x4 Checkerboard hashing
2h		16x4 Checkerboard hashing
3h		Reserved

Programming Notes
8x4 hashing preferred for when msaa enabled

GT_MODE - GT Mode Register

	8	**Reserved** Project: CHV, BSW Access: r/w Format: PBC
	7	**WIZ Hashing Mode** Project: CHV, BSW Access: r/w Format: U1 This field configures the Hashing mode in Windower. This field is don't care if the Hashing Disable bit is set. The WIZ Hashing Mode High Bit field is combined with this field to enable additional modes. (Project: CHV, BSW)
	6:3	**Reserved** Access: r/w Format: PBC
	2	**Reserved** Access: r/w Format: PBC
	1:0	**Reserved** Access: r/w Format: PBC

GTSCRATCH1

GTSCRATCH1 - GTSCRATCH1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F100h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH2

GTSCRATCH2 - GTSCRATCH2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F104h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH3

GTSCRATCH3 - GTSCRATCH3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F108h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH4

GTSCRATCH4 - GTSCRATCH4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F10Ch

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH5

GTSCRATCH5 - GTSCRATCH5	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F110h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH6

GTSCRATCH6 - GTSCRATCH6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F114h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH7

GTSCRATCH7 - GTSCRATCH7	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F118h

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCH8

GTSCRATCH8 - GTSCRATCH8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F11Ch

These (8 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**GT_Scratchpad** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

GTSCRATCHPAD0

GTSCRATCHPAD0 - GTSCRATCHPAD0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130040h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch0** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD1

GTSCRATCHPAD1 - GTSCRATCHPAD1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130044h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch1** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD2

GTSCRATCHPAD2 - GTSCRATCHPAD2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130048h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch2** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD3

GTSCRATCHPAD3 - GTSCRATCHPAD3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	13004Ch

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch3** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD4

GTSCRATCHPAD4 - GTSCRATCHPAD4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130050h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch4** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD5

GTSCRATCHPAD5 - GTSCRATCHPAD5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130054h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch5** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD6

GTSCRATCHPAD6 - GTSCRATCHPAD6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	130058h

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch6** Default Value: 00000000h Access: R/W GT scratchpad register.

GTSCRATCHPAD7

GTSCRATCHPAD7 - GTSCRATCHPAD7

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	13005Ch

GT scratchpad registers.
Scratchpad register can be R/W by both driver and GT.
One usage is that GT updates those registers before going to RC6 and reads them on RC6 exit.
Write-able and read-able from GT.
Reads do NOT rely on the register dispatch path, as dependencies could result.
Read-able and write-able from IA.
New for CHV, BSW.

DWord	Bit	Description
0	31:0	**GT_Scratch7** Default Value: 00000000h Access: R/W GT scratchpad register.

GTT Cache Enable

GTT_CACHE_EN - GTT Cache Enable

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	04024h

Enable GTT Cache for respective client(s), A0: Must program/observed this to all 0 due to Big Pages Bug 1898112

DWord	Bit	Description
0	31:0	**GTT Cache Enable for CS** Access: R/W Enable GTT Caching for all the client(s) below: 31: BLIT Engine (overrides individual enables of the units) 30: VEBX Engine (overrides individual enables of the units) 29: MFX Engine (overrides individual enables of the units) 28: GFX Engine (overrides individual enables of the units) 27-15: Reserved 14: VMCunit 13: VLFunit 12: BLBunit 11: VFWunit 10: VEOunit 9: HIZunit 8: RCZunit 7: RCCunit 6: ISCunit 5: DCunit 4: MTunit 3: SOLunit 2: VFunit 1: RSunit 0: CSunit

Value	Name	Project
00000000h	**[Default]**	CHV, BSW

GTTMMADR_LSB

GTTMMADR_LSB - GTTMMADR_LSB

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000004
Size (in bits):	32
Address:	00010h

Gfx Memory Mapped Address Range. This is the base
address for all memory mapped registers and GTT table.
This register requests allocation for the combined Graphics Translation Table Modification Range and Memory Mapped Range. For the Global GTT, this range is defined as a memory BAR in graphics device config space. It is an alias into which software is required to write Page Table Entry values (PTEs). Software may read PTE values from the global Graphics Translation Table (GTT). PTEs cannot be written directly into the global GTT memory area.
GTTMMADR comprises of 2 regions, GTT and MMADR. Prior to this feature it had a max size of 4M with 2M MMADR + {0,1M, 2M} GTT. Now GTTMMADR will have a max size of 16M.GTTMMADR = 2M MMADR + 6M rsvd + {0, 2M, 4M, 8M} GTT.GTTMMADR will be 16M aligned.

DWord	Bit	Description
0	31:24	**MBA_LSB** Default Value: 00h Access: R/W Memory Base Address (MBA): Set by the OS, these bits correspond to address signals [35:24].16MB combined for MMIO and Global GTT table aperture (2MB for MMIO, 6MB reserved and 8 MB for GTT).
	23:4	**Reserved** Default Value: 00000h Access: RO RSVD:Hardwired to 0 to indicate at least 4MB address range.
	3	**Reserved** Default Value: 0b Access: RO Prefetchable Memory (PREFMEM): Hardwired to 0to prevent prefetching.
	2:1	**MEMTYP** Default Value: 10b Access: RO Memory Type (MEMTYP):

GTTMMADR_LSB - GTTMMADR_LSB		
		00: To indicate 32 bit base address 01: Reserved 10: To indicate 64 bit base address 11: Reserved
	0	**Reserved**

GTTMMADR_MSB

GTTMMADR_MSB - GTTMMADR_MSB

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00014h

Gfx Memory Mapped Address Range. This is the base
address for all memory mapped registers and GTT table.
This register requests allocation for the combined Graphics Translation Table Modification Range and Memory Mapped Range. For the Global GTT, this range is defined as a memory BAR in graphics device config space. It is an alias into which software is required to write Page Table Entry values (PTEs). Software may read PTE values from the global Graphics Translation Table (GTT). PTEs cannot be written directly into the global GTT memory area.
GTTMMADR comprises of 2 regions, GTT and MMADR. Prior to this feature it had a max size of 4M with 2M MMADR + {0,1M, 2M} GTT. Now GTTMMADR will have a max size of 16M.GTTMMADR = 2M MMADR + 6M rsvd + {0, 2M, 4M, 8M} GTT.GTTMMADR will be 16M aligned.

DWord	Bit	Description
0	31:4	**MBA_MSB28** Default Value: 0000000h Access: R/W Description / Project: This field must be set to 0 since addressing above 64GB is not supported. / Made them spare RW bits. / CHV, BSW
	3:0	**MBA_MSB** Default Value: 0h Access: R/W Memory Base Address (MBA): Set by the OS, these bits correspond to address signals [35:24].16MB combined for MMIO and Global GTT table aperture (2MB for MMIO, 6MB reserved and 8 MB for GTT).

GU_CTL0

GU_CTL0 - GU_CTL0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000070
Size (in bits):	32
Address:	182030h

Gunit Configuration bits.

DWord	Bit	Description
0	31	**GT_IOSFSB_P_READ_POLICY** Default Value: 0b Access: R/W During an SOIx flow, the Punit must read Gunit registers. 0 (Default) : Allow IOSF SB P reads. Note : Punit knows when appropriate times are to issue IOSF SB P reads. 1 : Abort IOSF SB P reads. Note : This default is the opposite of the IOSF SB polarity bit 12_0008h[31]. Note : For CHV, BSW, this policy register will applies to new PM endpoint (which includes the new PM endpoint).
	30:16	**SPARE15** Default Value: 0000h Access: R/W Spare register bits for future use
	15	**SB_Doorbellprevention** Default Value: 0b Access: R/W 0 (default) : IOSF P to IOSF SB doorbell registers can be accesses by IOSF SB and IOSF SBp. Note : This should never occur during normal operation. 1 : IOSF SB and IOSF SBp accesses targeting the IOSF P to IOSF SB doorbell registers are aborted.
	14	**SPARE1** Default Value: 0b Access: R/W Spare register bits for future use
	13	**Reserved**

GU_CTL0 - GU_CTL0

12	**CMDperCLK_ECO** Default Value: 0b Access: R/W 0(default): GSA command per clock supported. 1: A GAM write followed by a read command cannot dispatch in consecutive clocked.
11	**Reserved**
10:9	**SPARE2** Default Value: 00b Access: R/W Spare bits for future use
8	**RESERVED** Default Value: 0b Access: RO Reserved
7	**IOSFP_DCR_Policy** Default Value: 0b Access: R/W IOSF Primary Data Credit Policy. 0 (default) : IOSF Primary data credits reflect data buffer depth. 1: IOSF Primary infinite data credits for both posted and non-posted data is advertised. Credit accounting (and flow control) will rely on cmd credits only.
6:4	**IOSFP_PCRD** Default Value: 111b Access: R/W IOSF Primary Credit AdvertisementNumber of posted command (and data credits/4) will advertize.
3	**SPARE3** Default Value: 0b Access: R/W Spare bits for future use
2	**Reserved**
1	**Reserved**
0	**PFI_Snoop_Override** Default Value: 0b

GU_CTL0 - GU_CTL0

Access:	R/W

0 : Observe GAM snoop/non-snoop indications.
1 : Force all GAM requests to snoop.

Hardware Status Mask Register

HWSTAM - Hardware Status Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0xFFFFFFFF
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1
Address:	02098h
Address:	12098h-1209Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VCSUNIT0
Address:	1A098h-1A09Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VECSUNIT
Address:	1C098h-1C09Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_VCSUNIT1
Address:	22098h-2209Bh
Name:	Hardware Status Mask Register
ShortName:	HWSTAM_BCSUNIT

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are mask bits that prevent the corresponding bits in the Interrupt Status Register from generating a Hardware Status Write (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

DWord	Bit	Description
0	31:0	**Hardware Status Mask Register Value** Default Value: FFFFFFFFh Format: Array of Masks Refer to the Interrupt Control Register section for bit definitions. Reserved bits are RO.

Hardware Status Page Address Register

HWS_PGA - Hardware Status Page Address Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Address:	02080h-02083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_RCSUNIT

Address:	12080h-12083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VCSUNIT0

Address:	1A080h-1A083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VECSUNIT

Address:	1C080h-1C083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_VCSUNIT1

Address:	22080h-22083h
Name:	Hardware Status Page Address Register
ShortName:	HWS_PGA_BCSUNIT

Description	Project
This register is used to program the 4 KB-aligned System Memory address of the Hardware Status Page used to report hardware status into (typically cacheable) System Memory.	
The address in this register is translated using the Global GTT in memory. The mapping type of the GTT entry determines the snoop nature of the transaction to memory.	CHV, BSW

DWord	Bit	Description
0	31:12	**Address** Format: GraphicsAddress[31:12] This field is used by SW to specify Bits 31:12 of the 4 KB-aligned System Memory address of the 4 KB page known as the Hardware Status Page. The Global GTT is used to map this page from the graphics virtual address to physical address.

HWS_PGA - Hardware Status Page Address Register

		Programming Notes If the Per-Process Virtual Address Space and Exec List Enable bit is set, HW requires that the status page is programmed to allow for the context switch status to be reported.
	11:0	**Reserved** Format: MBZ

HCP CABAC Status

HCP_CABAC_STATUS - HCP CABAC Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Size (in bits):	32
Trusted Type:	1
Address:	1E904h

HCP CABAC status.

DWord	Bit	Description
0	31:12	**Reserved** Format: MBZ
	11	**Temporal Direction Motion Vector Out-of-Bound Error** Default Value: 0 Access: RO Format: U1 This flag indicates motion vectors calculated from the Temporal Direct Motion vector is larger than the allowed range.
	10:7	**Reserved** Format: MBZ
	6	**Motion Vector Delta SE** Default Value: 0 Access: RO Format: U1 This flag indicates out-of-bound motion vector delta SEs coded in the bit-stream.
	5	**Delta QP SE** Default Value: 0 Access: RO Format: U1 This flag indicates leading-one overflow during CABAC decode of cu_qp_delta_abs.
	4	**Residual Error** Default Value: 0 Access: RO

HCP_CABAC_STATUS - HCP CABAC Status

		Format: U1 This flag indicates out-of-bound absolute coefficient level SEs coded in the bit-stream.
	3	**Slice and Error** Default Value: 0 Access: RO Format: U1 This flag indicates a pre-mature end to the slice or an inconsistent end of slice on the last Ctb of a slice.
	2:1	**Reserved** Format: MBZ
	0	**Ctb Concealment Flag** Default Value: 0 Access: RO Format: U1 Each pulse from this flag indicates one Ctb is concealed by the HCP.

HCP Decode Status

HCP_DEC_STATUS - HCP Decode Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E900h

HCP Decode status.

DWord	Bit	Description
0	31:18	**Number of Ctbs Concealed** Default Value: 0 Format: U14 This 16-bit field indicates the number of Ctbs concealed during the decoding of the current frame. This field is cleared with the HCP_PIPE_MODE_SELECT command.
	17	**Frame Dec Active** Default Value: 0 Format: U1 This flag indicates that the decoder hardware is actively decoding a picture.
	16	**Indirect Bitstream ObjectAccess Upper Bound Error** Default Value: 0 Format: U1 This flag indicates that the upper bound bit-stream address was reached.
	15:0	**Bit-stream Error Flags** Default Value: 0 Format: U16 This 16-bit field indicates the number of bit stream errors detected for each bit field indicated in the CABAC Status register.

HCP Picture Checksum cIdx0

HCP_PICTURE_CHECKSUM_CIDX0 - HCP Picture Checksum cIdx0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E91Ch

- The HCP Picture Checksum cIdx0 register reports the 32-bit unsigned picture checksum for cIdx=0 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx0** Default Value: 0 Format: U32

HCP Picture Checksum cIdx1

HCP_PICTURE_CHECKSUM_CIDX1 - HCP Picture Checksum cIdx1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E920h

- The HCP Picture Checksum cIdx1 register reports the 32-bit unsigned picture checksum for cIdx=1 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx1** Default Value: 0 Format: U32

HCP Picture Checksum cIdx2

HCP_PICTURE_CHECKSUM_CIDX2 - HCP Picture Checksum cIdx2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1E924h

- The HCP Picture Checksum cIdx2 register reports the 32-bit unsigned picture checksum for cIdx=2 calculated by the HCP hardware and whose algorithm is defined in Annex D of the HEVC standard specification.
- This calculated value is updated at the end of the frame.

DWord	Bit	Description
0	31:0	**Picture checksum cIdx2** Default Value: 0 Format: U32

HDR

HDR - HDR

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0000Ch

Header Type

DWord	Bit	Description
0	31:24	**RESERVED** Default Value: 00h Access: RO Reserved
	23	**MULTI_FUNCTION_STATUS** Default Value: 0b Access: RO MFUNC: Integrated graphics is a single function
	22:16	**HEADER_CODE** Default Value: 00h Access: RO HDR: Indicates a type 0 configuration space header format
	15:0	**RESERVED** Default Value: 0000h Access: RO Reserved

HEVC Local APIC Retry Vector

HEVC_LAPIC_RETRY_VECT - HEVC Local APIC Retry Vector

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0D594h
Valid Projects:	CHV, BSW

Holds the 4 last retry interrupt vectors. Retries are errors in hardware and are not expected. No interrupt is actually retried, and the interrupt stimulus will be lost if a retry occurs. The system will hang eventually. A 2-bit counter (starting at reset value of 0) is used to point to the slot/byte location from which to load the next retry vector (into the 4 available slots) in sequence. This means if a 5th retry vector shows up, it will be loaded into slot 0 again (as the counter wraps around), over-writing the retry vector which existed there in slot_0.

DWord	Bit	Description
0	31:24	**Vector Slot 3** Format: U8
	23:16	**Vector Slot 2** Format: U8
	15:8	**Vector Slot 1** Format: U8
	7:0	**Vector Slot 0** Format: U8

HS Invocation Counter

HS_INVOCATION_COUNT - HS Invocation Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02300h

This register stores the number of patch objects processed by the HS unit. E.g., A PATCHLIST_2 topology with 6 vertices would cause this counter to increment by 3 (there are 3 2-vertex patch objects in that topology).This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**HS Invocation Count UDW** Number of patch objects processed by the HS stage. Updated only when HS Enable and HS Statistics Enable are set in 3DSTATE_HS
	31:0	**HS Invocation Count LDW** Number of patch objects processed by the HS stage. Updated only when HS Enable and HS Statistics Enable are set in 3DSTATE_HS

IA32 MTRR FIX4K_C0000 High

MTRR_FIX4K_C0000_H - IA32 MTRR FIX4K_C0000 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F13Ch	
Fixed MTRR to identify (C0000-C8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_C0000 Low

MTRR_FIX4K_C0000_L - IA32 MTRR FIX4K_C0000 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F138h

Fixed MTRR to identify (C0000-C8000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_C8000 High

MTRR_FIX4K_C8000_H - IA32 MTRR FIX4K_C8000 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F144h	
Fixed MTRR to identify (C8000-D0000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_C8000 Low

MTRR_FIX4K_C8000_L - IA32 MTRR FIX4K_C8000 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F140h

Fixed MTRR to identify (C8000-D0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_D0000 High

MTRR_FIX4K_D0000_H - IA32 MTRR FIX4K_D0000 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F14Ch	
Fixed MTRR to identify (D0000-D8000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_D0000 Low

MTRR_FIX4K_D0000_L - IA32 MTRR FIX4K_D0000 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F148h

Fixed MTRR to identify (D0000-D8000h)

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_D8000 High

MTRR_FIX4K_D8000_H - IA32 MTRR FIX4K_D8000 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F154h	
Fixed MTRR to identify (D8000-E0000h)		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_D8000 Low

MTRR_FIX4K_D8000_L - IA32 MTRR FIX4K_D8000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F150h

Fixed MTRR to identify (D8000-E0000h)

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_E0000 High

MTRR_FIX4K_E0000_H - IA32 MTRR FIX4K_E0000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F15Ch

Fixed MTRR to identify (E0000-E8000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_E0000 Low

MTRR_FIX4K_E0000_L - IA32 MTRR FIX4K_E0000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F158h

Fixed MTRR to identify (E0000-E8000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_E8000 High

MTRR_FIX4K_E8000_H - IA32 MTRR FIX4K_E8000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F164h

Fixed MTRR to identify (E8000-F0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_E8000 Low

MTRR_FIX4K_E8000_L - IA32 MTRR FIX4K_E8000 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F160h

Fixed MTRR to identify (E8000-F0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_F0000 High

MTRR_FIX4K_F0000_H - IA32 MTRR FIX4K_F0000 High		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F16Ch	
Fixed MTRR to identify (F0000-F8000h).		
DWord	**Bit**	**Description**
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_F0000 Low

MTRR_FIX4K_F0000_L - IA32 MTRR FIX4K_F0000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F168h

Fixed MTRR to identify (F0000-F8000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX4K_F8000 High

MTRR_FIX4K_F8000_H - IA32 MTRR FIX4K_F8000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F174h

Fixed MTRR to identify (F8000-100000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX4K_F8000 Low

MTRR_FIX4K_F8000_L - IA32 MTRR FIX4K_F8000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F170h

Fixed MTRR to identify (F8000-100000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX16K_80000 High

MTRR_FIX16K_80000_H - IA32 MTRR FIX16K_80000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F12Ch

Fixed MTRR to identify 512K-768K of the main memory (80000-A0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX16K_80000 Low

MTRR_FIX16K_80000_L - IA32 MTRR FIX16K_80000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F128h

Fixed MTRR to identify 512K-768K of the main memory (80000-A0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#.0

IA32 MTRR FIX16K_A0000 High

MTRR_FIX16K_A0000_H - IA32 MTRR FIX16K_A0000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F134h

Fixed MTRR to identify 768K-1024K of the main memory (A0000-C0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX16K_A0000 Low

MTRR_FIX16K_A0000_L - IA32 MTRR FIX16K_A0000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F130h

Fixed MTRR to identify 768K-1024K of the main memory (A0000-C0000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR FIX64K_00000 High

MTRR_FIX64K_00000_H - IA32 MTRR FIX64K_00000 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F124h

Fixed MTRR to identify 0-512K of the main memory (0-80000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[63:56]: Identifies the memory type 00h-FFh of range#7. Bit[55:48]: Identifies the memory type 00h-FFh of range#6. Bit[47:40]: Identifies the memory type 00h-FFh of range#5. Bit[39:32]: Identifies the memory type 00h-FFh of range#4.

IA32 MTRR FIX64K_00000 Low

MTRR_FIX64K_00000_L - IA32 MTRR FIX64K_00000 Low	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F120h

Fixed MTRR to identify 0-512K of the main memory (0-80000h).

DWord	Bit	Description
0	31:0	**Range0 to Range7 Memory Type** Default Value: 00000000h Access: R/W Bit[31:24]: Identifies the memory type 00h-FFh of range#3. Bit[23:16]: Identifies the memory type 00h-FFh of range#2. Bit[15:8]: Identifies the memory type 00h-FFh of range#1. Bit[7:0]: Identifies the memory type 00h-FFh of range#0.

IA32 MTRR PHYSBASE0 High

MTRR_PHYSBASE0_H - IA32 MTRR PHYSBASE0 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F184h

Variable MTRR0

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE0 Low

MTRR_PHYSBASE0_L - IA32 MTRR PHYSBASE0 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F180h

Variable MTRR0

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE1 High

MTRR_PHYSBASE1_H - IA32 MTRR PHYSBASE1 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F194h

Variable MTRR1

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE1 Low

MTRR_PHYSBASE1_L - IA32 MTRR PHYSBASE1 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F190h

Variable MTRR1

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE2 High

MTRR_PHYSBASE2_H - IA32 MTRR PHYSBASE2 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1A4h

Variable MTRR2

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE2 Low

MTRR_PHYSBASE2_L - IA32 MTRR PHYSBASE2 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1A0h

Variable MTRR2

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE3 High

MTRR_PHYSBASE3_H - IA32 MTRR PHYSBASE3 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1B4h

Variable MTRR3

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE3 Low

MTRR_PHYSBASE3_L - IA32 MTRR PHYSBASE3 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1B0h

Variable MTRR3

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE4 High

MTRR_PHYSBASE4_H - IA32 MTRR PHYSBASE4 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1C4h

Variable MTRR4

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE4 Low

MTRR_PHYSBASE4_L - IA32 MTRR PHYSBASE4 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1C0h

Variable MTRR4

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh

IA32 MTRR PHYSBASE5 High

MTRR_PHYSBASE5_H - IA32 MTRR PHYSBASE5 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1D4h

Variable MTRR5

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE5 Low

MTRR_PHYSBASE5_L - IA32 MTRR PHYSBASE5 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1D0h

Variable MTRR5

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE6 High

MTRR_PHYSBASE6_H - IA32 MTRR PHYSBASE6 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1E4h

Variable MTRR6

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE6 Low

MTRR_PHYSBASE6_L - IA32 MTRR PHYSBASE6 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1E0h

Variable MTRR6

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE7 High

MTRR_PHYSBASE7_H - IA32 MTRR PHYSBASE7 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F4h

Variable MTRR7

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE7 Low

MTRR_PHYSBASE7_L - IA32 MTRR PHYSBASE7 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F0h

Variable MTRR7

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE8 High

MTRR_PHYSBASE8_H - IA32 MTRR PHYSBASE8 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F204h

Variable MTRR8

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE8 Low

MTRR_PHYSBASE8_L - IA32 MTRR PHYSBASE8 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F200h

Variable MTRR8

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSBASE9 High

MTRR_PHYSBASE9_H - IA32 MTRR PHYSBASE9 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F214h

Variable MTRR9

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysBase** Default Value: 0000000b Access: R/W Physical Base address[38:32] of the variable MTRR.

IA32 MTRR PHYSBASE9 Low

MTRR_PHYSBASE9_L - IA32 MTRR PHYSBASE9 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F210h

Variable MTRR9

DWord	Bit	Description
0	31:12	**PhysBase** Default Value: 00000h Access: R/W Physical Base address[31:0] of the variable MTRR.
	11:8	**Reserved** Default Value: 0000b Access: RO
	7:0	**Memory Type** Default Value: 00h Access: R/W Identifies the memory type 00h-FFh.

IA32 MTRR PHYSMASK0 High

MTRR_PHYSMASK0_H - IA32 MTRR PHYSMASK0 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F18Ch

Variable MTRR0

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK0 Low

MTRR_PHYSMASK0_L - IA32 MTRR PHYSMASK0 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F188h

Variable MTRR0

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK1 High

MTRR_PHYSMASK1_H - IA32 MTRR PHYSMASK1 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F19Ch

Variable MTRR1

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK1 Low

MTRR_PHYSMASK1_L - IA32 MTRR PHYSMASK1 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F198h

Variable MTRR1

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK2 High

MTRR_PHYSMASK2_H - IA32 MTRR PHYSMASK2 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1ACh

Variable MTRR2

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK2 Low

MTRR_PHYSMASK2_L - IA32 MTRR PHYSMASK2 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1A8h

Variable MTRR2

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK3 High

MTRR_PHYSMASK3_H - IA32 MTRR PHYSMASK3 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1BCh

Variable MTRR3

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK3 Low

MTRR_PHYSMASK3_L - IA32 MTRR PHYSMASK3 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1B8h

Variable MTRR3

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK4 High

MTRR_PHYSMASK4_H - IA32 MTRR PHYSMASK4 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1CCh

Variable MTRR4

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK4 Low

MTRR_PHYSMASK4_L - IA32 MTRR PHYSMASK4 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1C8h

Variable MTRR4

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK5 High

MTRR_PHYSMASK5_H - IA32 MTRR PHYSMASK5 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1DCh

Variable MTRR5

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK5 Low

MTRR_PHYSMASK5_L - IA32 MTRR PHYSMASK5 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1D8h

Variable MTRR5

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK6 High

MTRR_PHYSMASK6_H - IA32 MTRR PHYSMASK6 High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1ECh

Variable MTRR6

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK6 Low

MTRR_PHYSMASK6_L - IA32 MTRR PHYSMASK6 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1E8h

Variable MTRR6

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK7 High

MTRR_PHYSMASK7_H - IA32 MTRR PHYSMASK7 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1FCh

Variable MTRR7

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK7 Low

MTRR_PHYSMASK7_L - IA32 MTRR PHYSMASK7 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F1F8h

Variable MTRR7

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK8 High

MTRR_PHYSMASK8_H - IA32 MTRR PHYSMASK8 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F20Ch

Variable MTRR8

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK8 Low

MTRR_PHYSMASK8_L - IA32 MTRR PHYSMASK8 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F208h

Variable MTRR8

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA32 MTRR PHYSMASK9 High

MTRR_PHYSMASK9_H - IA32 MTRR PHYSMASK9 High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F21Ch

Variable MTRR9

DWord	Bit	Description
0	31:7	**Reserved** Default Value: 00000000000000000000000000b Access: RO
	6:0	**PhysMask** Default Value: 0000000b Access: R/W Physical MASK for the address[38:32] of the variable MTRR.

IA32 MTRR PHYSMASK9 Low

MTRR_PHYSMASK9_L - IA32 MTRR PHYSMASK9 Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F218h

Variable MTRR9

DWord	Bit	Description
0	31:12	**PhysMask** Default Value: 00000h Access: R/W Physical MASK for the address[31:0] of the variable MTRR.
	11	**Valid** Default Value: 0b Access: R/W Valid bit showing that MTRR decode is active.
	10:0	**Reserved** Default Value: 00000000000b Access: RO

IA Vertices Count

IA_VERTICES_COUNT - IA Vertices Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02310h
Valid Projects:	

This register stores the count of vertices processed by VF. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**IA Vertices Count Report UDW** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)
	31:0	**IA Vertices Count Report LDW** Total number of vertices fetched by the VF stage. This count is updated for every input vertex as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)

IDLE Messaging Register for Blitter Engine

MSG_IDLE_BCS - IDLE Messaging Register for Blitter Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	0800Ch

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:6	**Reserved** Access: RO
	5	**Flush and Block Acknowledgement** Access: R/W Flush and Block Acknowledgement 1'b0 : Not flushed and blocked <default> 1'b1 : Unit has flushed and blocked its pipeline
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received.
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated

MSG_IDLE_BCS - IDLE Messaging Register for Blitter Engine		
		Bit[1] C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] Idle Indication 1'b0 : Pipe is busy <default> 1'b1 : Pipe is idle ** See "Valid Combinations for Idle Messaging" Table

IDLE Messaging Register for Media0 Engine

MSG_IDLE_VCS0 - IDLE Messaging Register for Media0 Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08004h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:6	**Reserved** Access: RO
	5	**Flush and Block Acknowledgement** Access: R/W Flush and Block Acknowledgement 1'b0 : Not flushed and blocked <default> 1'b1 : Unit has flushed and blocked its pipeline
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received.
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated Bit[1]

MSG_IDLE_VCS0 - IDLE Messaging Register for Media0 Engine

		C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] IDLE Indication - Media done 1'b0 : Pipe is busy <default> 1'b1 : Pipe is idle Refer to bits A024[10:9] for different Media turbo scenarios. ** See "Valid Combinations for Idle Messaging" Table

IDLE Messaging Register for Media1 Engine

MSG_IDLE_VCS1 - IDLE Messaging Register for Media1 Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08008h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:6	**Reserved** Access: RO
	5	**Flush and Block Acknowledgement** Access: R/W Flush and Block Acknowledgement 1'b0 : Not flushed and blocked <default> 1'b1 : Unit has flushed and blocked its pipeline
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received.
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated Bit[1]

MSG_IDLE_VCS1 - IDLE Messaging Register for Media1 Engine		
		C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] Idle Indication 1'b0 : Pipe is busy <default> 1'b1 : Pipe is idle ** See "Valid Combinations for Idle Messaging" Table

IDLE Messaging Register for Render Engine

MSG_IDLE_CS - IDLE Messaging Register for Render Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	08000h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.
NOTE - GPGPU_ACTIVE and MEDIA_ACTIVE are context saved and restored; the other bits are cleared out before saving context

DWord	Bit	Description
0	31:16	**Context Save Mask** Access: R/W When context save is in progress, mask is forced to particular value to save off messages that need to be retained across an RC6 event. Currently for this register, the following fields are context saved and restored for an RC6 event: [7] GPGPU_ACTIVE [6] MEDIA_ACTIVE
	15:8	**Reserved** Access: RO
	7	**GPGPU Active Load Indication** Access: R/W Active Load Indication - bits[7:6] of this register 2'b00 : Render load is being executed <default> 2'b01 : Media load is being executed 2'b10 : GPGPU load is being executed 2'b11 : Undefined gpmunit self-clears this bit upon sampling.
	6	**Media Active Load Indication** Access: R/W Active Load Indication - bits[7:6] of this register 2'b00 : Render load is being executed <default> 2'b01 : Media load is being executed 2'b10 : GPGPU load is being executed

MSG_IDLE_CS - IDLE Messaging Register for Render Engine

		2'b11 : Undefined gpmunit self-clears this bit upon sampling.
	5	**Flush and Block Acknowledgement** Access: R/W Flush and Block Acknowledgement 1'b0 : Not flushed and blocked <default> 1'b1 : Unit has flushed and blocked its pipeline
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received. Also, Refer to the table in the description for bits A024[10:9]
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated Bit[1] C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] Idle Indication 1'b0 : Pipe is busy <default> 1'b1 : Pipe is idle ** See "Valid Combinations for Idle Messaging" Table Also, Refer to the table in the description for bits A024[10:9]

IDLE Messaging Register for VEBox

MSG_IDLE_VECS - IDLE Messaging Register for VEBox

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08010h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:6	**Reserved** Access: RO
	5	**Flush and Block Acknowledgement** Access: R/W Flush and Block Acknowledgement 1'b0 : Not flushed and blocked <default> 1'b1 : Unit has flushed and blocked its pipeline
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received.
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated Bit[1]

MSG_IDLE_VECS - IDLE Messaging Register for VEBox		
		C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] Idle Indication 1'b0 : Pipe is busy <default> 1'b1 : Pipe is idle ** See "Valid Combinations for Idle Messaging" Table

IDLE Messaging Register for Wi-Di

MSG_IDLE_WIN - IDLE Messaging Register for Wi-Di

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08014h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:5	**Reserved** Access: RO
	4	**Preparation for Reset Acknowledgement** Access: R/W Go Acknowledgement 1'b0 : Go=0 Ack <default> 1'b1 : Go=1 Ack Requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received.
	3:0	**Idle Messaging** Access: R/W Idle Messaging Bit[3] Secondary Pipe Clock Gating 1'b0 : Secondary pipe clock must be on <default> 1'b1 : Secondary pipe clock may be gated Bit[2] Primary Pipe Clock Gating 1'b0 : Primary pipe clock must be on <default> 1'b1 : Primary pipe clock may be gated Bit[1] C6 Allowed 1'b0 : Do not allow GT to enter C6 <default> 1'b1 : GT may enter C6 Bit[0] Idle Indication 1'b0 : Pipe is busy <default>

MSG_IDLE_WIN - IDLE Messaging Register for Wi-Di		
		1'b1 : Pipe is idle ** See "Valid Combinations for Idle Messaging" Table

Idle Switch Delay

IDLEDLY - Idle Switch Delay	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0223Ch
Address:	1223Ch-1223Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VCSUNIT0
Address:	1A23Ch-1A23Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VECSUNIT
Address:	1C23Ch-1C23Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_VCSUNIT1
Address:	2223Ch-2223Fh
Name:	Idle Switch Delay
ShortName:	IDLEDLY_BCSUNIT

The IDLEDLY register contains an Idle Delay field which specifies the minimum number of microseconds allowed for command streamer to wait before a context is switched out leading to IDLE state in Execlist mode, i.e following this context switch there is no active element available in HW to execute.
A default value of 0, means that by default, there is no restriction to wait on a context switch leading to IDLE. This register has no significance when Execlists are not enabled.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:0	**IDLE Delay** Project: All Format: U21 Minimum number of micro-seconds allowed

Indirect Context Pointer

INDIRECT_CTX - Indirect Context Pointer

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Address:	021C4h-021C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_RCSUNIT

Address:	121C4h-121C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VCSUNIT0

Address:	1A1C4h-1A1C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VECSUNIT

Address:	1C1C4h-1C1C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_VCSUNIT1

Address:	221C4h-221C7h
Name:	Indirect Context Pointer
ShortName:	INDIRECT_CTX_BCSUNIT

This register is used to program the indirect address to be executed between CS and SVG engine context if enabled. This will only get executed due to regular context save/restore and not during power restore. This register is part of the execution list context and will be executed per context. Only supported if execution list is enabled. There is no preempting workloads within this context.

Programming Notes	Source
The following commands are not supported within Render CS indirect context: **Command Name** MI_WAIT_FOR_EVENT MI_SEMAPHORE_SIGNAL MI_ARB_CHECK MI_RS_CONTROL MI_REPORT_HEAD	RenderCS

INDIRECT_CTX - Indirect Context Pointer

MI_URB_ATOMIC_ALLOC
MI_SUSPEND_FLUSH
MI_TOPOLOGY_FILTER
MI_RS_CONTEXT
MI_SET_CONTEXT
MI_URB_CLEAR
MI_SEMAPHORE_WAIT in Memory Poll Mode is not supported.
MI_BATCH_BUFFER_START
MI_CONDITIONAL_BATCH_BUFFER_END
MEDIA_OBJECT_WALKER
GPGPU_WALKER
3DPRIMITIVE
3DSTATE_BINDING_TABLE_POINTERS_VS
3DSTATE_BINDING_TABLE_POINTERS_HS
3DSTATE_BINDING_TABLE_POINTERS_DS
3DSTATE_BINDING_TABLE_POINTERS_GS
3DSTATE_BINDING_TABLE_POINTERS_PS
3DSTATE_GATHER_CONSTANT_VS
3DSTATE_GATHER_CONSTANT_GS
3DSTATE_GATHER_CONSTANT_HS
3DSTATE_GATHER_CONSTANT_DS
3DSTATE_GATHER_CONSTANT_PS
3DSTATE_DX9_CONSTANTF_VS
3DSTATE_DX9_CONSTANTF_HS
3DSTATE_DX9_CONSTANTF_DS
3DSTATE_DX9_CONSTANTF_GS
3DSTATE_DX9_CONSTANTF_PS
3DSTATE_DX9_CONSTANTI_VS
3DSTATE_DX9_CONSTANTI_HS
3DSTATE_DX9_CONSTANTI_DS
3DSTATE_DX9_CONSTANTI_GS
3DSTATE_DX9_CONSTANTI_PS
3DSTATE_DX9_CONSTANTB_VS
3DSTATE_DX9_CONSTANTB_HS
3DSTATE_DX9_CONSTANTB_DS

INDIRECT_CTX - Indirect Context Pointer

3DSTATE_DX9_CONSTANTB_GS		
3DSTATE_DX9_CONSTANTB_PS		
3DSTATE_DX9_LOCAL_VALID_VS		
3DSTATE_DX9_LOCAL_VALID_DS		
3DSTATE_DX9_LOCAL_VALID_HS		
3DSTATE_DX9_LOCAL_VALID_GS		
3DSTATE_DX9_LOCAL_VALID_PS		
3DSTATE_DX9_GENERATE_ACTIVE_VS		
3DSTATE_DX9_GENERATE_ACTIVE_HS		
3DSTATE_DX9_GENERATE_ACTIVE_DS		
3DSTATE_DX9_GENERATE_ACTIVE_GS		
3DSTATE_DX9_GENERATE_ACTIVE_PS		
3DSTATE_BINDING_TABLE_EDIT_VS		
3DSTATE_BINDING_TABLE_EDIT_GS		
3DSTATE_BINDING_TABLE_EDIT_HS		
3DSTATE_BINDING_TABLE_EDIT_DS		
3DSTATE_BINDING_TABLE_EDIT_PS		
3DSTATE_CONSTANT_VS		
3DSTATE_CONSTANT_GS		
3DSTATE_CONSTANT_PS		
3DSTATE_CONSTANT_HS		
3DSTATE_CONSTANT_DS		
MI_BATCH_BUFFER_END		

INDIRECT_CTX - Indirect Context Pointer

Workaround
Workaround: [Render CS Only][Execlist Mode of Scheduling]: SW must ensure arbitration is switched off while context restore is in progress for any given context. This is achieved by disabling arbitration by programming MI_ARB_ON_OFF to "Arbitration Disable" in RCS_INDIRECT_CTX buffer and by enabling back the arbitration by programming MI_ARB_ON_OFF to "Arbitration Enable" as the last command prior to MI_BATCH_END in the BB_PER_CTX_PTR buffer of every context submitted. Note that RCS_INDIRECT_CTX_OFFSET could be set to default value or any other legitimate value as per the programming notes of the register definition. Arbitration disable by programming MI_ARB_ON_OFF (Arbitration Disabled) in RCS_INDIRECT_CTX buffer. Arbitration enabled by programming MI_ARB_ON_OFF (Arbitration Enabled) as the last command prior to MI_BATCH_BUFFER_END in BB_PER_CTX_PTR buffer. Additional Note: This WA need not be applied when it is guaranteed for no preemption to occur during execution of GPGPU workload. Preemption of GPGPU workload can be avoided by Bracketing the GPGPU workload with MI_ARB_ON_OFF (Arbitration Disable) and MI_ARB_ON_OFF (Arbitration Enable) command. MI_ARB_ON_OFF is a privileged command and can only be programmed in ring buffer or in privileged batch buffer (batch buffer in GGTT memory). Pending execlist submitted must not trigger preemption of the ongoing GPGPU workload due to following reasons First context of the pending execlist submitted is not the same as the ongoing GPGPU context. Force restore bit set for the submitted pending execlist.

DWord	Bit	Description
0	31:6	**Indirect CS Context Address** Format: GraphicsAddress[31:6] Pointer to the Context in memory to be executed as a batch.
	5:0	**Size of Indirect CS Context** Format: U6 This is the size of the Indirect Context for CS. This size supports up to 63 cache lines worth of commands where a cache line is 64B. If programmed to zero then the indirect fetch of the CS context is disabled. Value: [0,63] Name:

Instruction Parser Mode Register

INSTPM - Instruction Parser Mode Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00004080 [CHV:B, CHV:C, CHV:K]
	0x00006080 [CHV:A]
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1
Address:	020C0h
Address:	120C0h-120C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VCSUNIT0
Address:	1A0C0h-1A0C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VECSUNIT
Address:	1C0C0h-1C0C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_VCSUNIT1
Address:	220C0h-220C3h
Name:	Instruction Parser Mode Register
ShortName:	INSTPM_BCSUNIT

The INSTPM register is used to control the operation of the Instruction Parser. Certain classes of instructions can be disabled (ignored) - often useful for detecting performance bottlenecks. Also, Synchronizing Flush operations can be initiated - useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes

- If an instruction type is disabled, the parser will read those instructions but not process them.
- Error checking will be performed even if the instruction is ignored.
- All Reserved bits are implemented.
- This Register is saved and restored as part of Context.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>31:16</td><td>Mask Bits
<table>
<tr><td>Format:</td><td>Mask[15:0]</td></tr>
</table>
Masks: These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.</td></tr>
<tr><td></td><td>15</td><td>Reserved
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Access:</td><td>RO</td></tr>
<tr><td>Format:</td><td>MBZ</td></tr>
</table></td></tr>
<tr><td></td><td>14</td><td>Replay Mode
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Format:</td><td>U1</td></tr>
</table>
This field controls the granularity of the replay mechanism when coming back into a previously preempted context.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>1h</td><td>Object Level Preemption [Default]</td><td>Object Level. Preemption is done on an Object Level Boundary in VF. Objects send down by VF are completely rendered. Pipeline is flushed before switching to the next context. On resubmission of the context VF starts parsing form the object where it got preempted last time.</td></tr>
<tr><td>0h</td><td></td><td>Drawcall Level. Pipeline is flushed before switching to the next context. Commands parsed are committed to completing before a context switch.</td></tr>
</table>
<table>
<tr><th>Programming Notes</th></tr>
<tr><td>This bit ust be set to 0 prior to any 3DPRIMITVE using trifan, polygon, lineloop or quadstrip topology.
This bit ust be set to 0 prior to any 3DPRIMITVE using linestrip_adjacency and 3dstate_GS.enable is set to 1.</td></tr>
</table></td></tr>
<tr><td></td><td>14:13</td><td>Reserved
<table>
<tr><td>Default Value:</td><td>11b</td></tr>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Format:</td><td>Must Be One</td></tr>
</table></td></tr>
<tr><td></td><td>13</td><td>Reserved
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Format:</td><td>Must Be One</td></tr>
</table></td></tr>
<tr><td></td><td>12</td><td>Reserved
<table>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
</table></td></tr>
<tr><td></td><td>11</td><td>CLFLUSH Toggle</td></tr>
</table>

DWord	Bit	Description
		Project: CHV, BSW Access: RO Format: U1 This bit changes polarity each time the MI_CLFLUSH command completes. This bit is Read Only.
	10	**Reserved**
	9:8	**Reserved** Project: CHV, BSW Format: MBZ
	7	**Force Sync Command Ordering** Default Value: 1b Project: CHV, BSW Format: Enable By default, driver/OS synchronization commands (MI_STORE_DATA_IMM, for instance) can execute out of order with respect to 3D state and 3D primitive commands. When set, this bit forces ordering of these commands. See section 3.2.2 for a list of these commands.
	6	**CONSTANT_BUFFER Address Offset Disable** Project: CHV, BSW Format: Disable When this bit is clear, the 3DSTATE_CONSTANT_* Buffers' Starting Address is used as a DynamicStateOffset. That is, it serves as an offset from the Dynamic State Base Address. Accesses will be subject to Dynamic State bounds checking. When this bit is set, the 3DSTATE_CONSTANT_* Buffers' Starting Address is used as a true GraphicsAddress (not an offset). No bounds checking will be performed during access.
	5	**Reserved** Project: CHV, BSW Format: MBZ
	4	**Reserved** Project: CHV, BSW
	3	**Media Instruction Disable** Project: CHV, BSW Format: U1 This bit instructs the Renderer instruction parser to parse and error-check Media instructions, but not execute them. Format = Disable

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td>2</td><td>3D Rendering Instruction Disable<table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>U1</td></tr></table>This bit instructs the Renderer instruction parser to parse and error-check 3D Rendering instructions, but not execute them. This bit must always be set by software if 3D State Instruction Disable is set. Setting this bit without setting 3D State Instruction Disable is allowed.
Format = Disable</td></tr>
<tr><td></td><td>1</td><td>3D State Instruction Disable<table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>Disable</td></tr></table></td></tr>
<tr><td></td><td>0</td><td>Texture Palette Load Instruction Disable<table><tr><td>Project:</td><td>CHV, BSW</td></tr><tr><td>Format:</td><td>U1</td></tr></table>This bit instructs the Renderer instruction parser to parse and error-check Texture Palette Load instructions, but not execute them. Format = Disable</td></tr>
</table>

Internal GAM State

INTSTATE - Internal GAM State			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	040C0h		
DWord		**Bit**	**Description**
0		31:0	**Reserved**

INTERRUPT LINE

INTRLINE - INTERRUPT LINE

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000100
Size (in bits):	16
Address:	0003Ch

3C - Interrupt. This register is programmed by SBIOS. It is not used by the graphics/display driver. This 8-bit register is used to communicate interrupt line routing information. It is read/write and must be implemented by the device. POST software will write the routing information into this register as it initializes and configures the system.
The value in this register tells which input of the system interrupt controller(s) the device's interrupt pin is connected to. The device itself does not use this value, rather it is used by device drivers and operating systems to determine priority and vector information.
3D - Interrupt. This register is programmed by SBIOS. It is not used by the graphics/display driver

DWord	Bit	Description
0	15:8	**INTERRUPT_PIN** Default Value: 01h Access: RO IPIN: Value indicates which interrupt pin this device uses. 01h: INTA
	7:0	**INTRLINE** Default Value: 00h Access: R/W ILIN: BIOS written value to communicate interrupt line routing information to the device driver. Used to communicate interrupt line routing information. POST software writes the routing information into this register as it initializes and configures the system. The value in this register indicates to which input of the system interrupt controller the device's interrupt pin is connected.

Interrupt Mask Register

IMR - Interrupt Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0xFFFFFFFF
Access:	R/W, RO
Size (in bits):	32
Address:	020A8h
Address:	120A8h-120ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VCSUNIT0
Address:	1A0A8h-1A0ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VECSUNIT
Address:	1C0A8h-1C0ABh
Name:	Interrupt Mask Register
ShortName:	IMR_VCSUNIT1
Address:	220A8h-220ABh
Name:	Interrupt Mask Register
ShortName:	IMR_BCSUNIT

The IMR register is used by software to control which Interrupt Status Register bits are masked or unmasked. Unmasked bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. Masked bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** Format: InterruptMask[32] Refer to the Interrupt Control Register section for bit definitions. This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR. Reserved bits in the Interrupt Control Register are RO.

Value	Name	Description
FFFF FFFFh	**[Default]**	
0h	Not Masked	Will be reported in the IIR
1h	Masked	Will not be reported in the IIR

IOBAR

IOBAR - IOBAR

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001
Size (in bits):	32
Address:	00020h

I/O Base Address. This is used only by SBIOS. This register is the base address for the MMIO_INDEX and MMIO_DATA registers.

This register provides the Base offset of the I/O registers within Device #2. Access to the 8Bs of IO space is allowed in PM state D0 when IO Enable (PCICMD bit 0) set. Access is disallowed :
* in PM states D1-D3 or
* if IO Enable is clear or
* if Device #2 is turned off or
* if Internal graphics is disabled thru the fuse or fuse override mechanisms.

Note that access to this IO BAR is independent of VGA functionality within Device #2.
If accesses to this IO bar is allowed then the GMCH claims all 8, 16 or 32 bit IO cycles from the CPU that falls within the 8B claimed.

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved
	15:6	**BASE_ADDRESS** Default Value: 0000h Access: R/W BA: Set by the OS, these bits correspond to address signals [15:6]. IOBAR is to be used for both GTLC register programming and GTT table programming. This is an indirect access method.
	5:1	**RESERVED** Default Value: 0h Access: RO
	0	**RESOURCE_TYPE_RTE** Default Value: 1b Access: RO Indicates a request for I/O space

L3 Bank Status

L3STAT - L3 Bank Status		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B128h	
L3 Status register		
DWord	**Bit**	**Description**
0	31	**L3 Fill Access Status bit** Access: RO This register is Hardware Set and Clear. Set condition: set when the first command is seen on LTCC-LTCD interface. Reset condition: reset when the first Pipeline Flush command is seen on the LTCC-LTCD interface. Reset condition: This Flag will be reset only if we have atleast 1 modified line in the cache written by DC client.
	30:0	**Reserved** Project: CHV, BSW Access: RO

L3CD Error Status register 1

L3CDERRST - L3CD Error Status register 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000080
Size (in bits):	32
Address:	0B1F0h

DWord	Bit	Description
0	31:25	**Reserved** Access: RO
	24	**Double bit ECC error detected** Default Value: 0b Access: R/W One Clear Indicates if bank detected a double bit ECC error. When ltcd_lbcf_ecc_2bit_err_valid is set.
	23:14	**Parity row address error** Default Value: 0000000000b Access: R/W One Clear Data array address which has parity B1. Report the data array address which has the Error. ltcd_lbcf_parity_err_rownum[9:0].Once set by HW, it can be cleared only by MMIO Write of 1 to this register bit 13. Driver needs to write 1 to clear this bit.
	13	**Parity Error Valid** Access: R/W One Clear Parity Error valid. Report the Parity Error. ltcd_lbcf_parity_err_valid. Once set by HW, it can be cleared only by MMIO Write of 1 to this register bit 13. Driver needs to write 1 to clear this bit. when ltcd_lbcf_parity_err_valid is asserted, lbcf generates interrupt to ltiseqsl lbcf_ltiseqsl_parity_intr.
	12:11	**Parity error bank number** Access: R/W One Clear bank number which has parity error. Report the bank no. which has the Error. ltcd_lbcf_parity_err_banknum[1:0].Once set by HW, it can be cleared only by MMIO Write of 1 to this register bit 13. Driver needs to write 1 to clear this bit.

L3CDERRST - L3CD Error Status register 1

	10:8	**Parity Error sub-bank no** Access: R/W One Clear Parity Error in sub bank: ltcd0_lbcf_parity_err_subanknum[2:0].Once set by HW, it can be cleared only by MMIO Write of 1 to this register bit 13. Driver needs to write 1 to clear this bit.
	7	**Parity report enable** Default Value: 1b Access: R/W Parity report enable (LCPRTYRPTEN): lbcf_csr_lc_parity_report_en. This is the parity reporting enable, by default it is enabled. When enabled parity is reported by ltcd to sarb. When disabled by driver, ltcd should not send out any parity error to SARB. Driver needs to write 1 to clear this bit.
	6:0	**Reserved** Access: RO

L3 Control Register

L3CNTLREG - L3 Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	07034h

Programming Notes	Project
The L3 allocation programming should assign all ways of the cache with no left over ways. Refer to L3 section for the recommended settings.	
Any L3 configuration change that reduces the data cache allocation when strong IA coherency is used requires the full flush of L3 prior to the programming update. An explicit or implicit flush of L3 (DC Flush) through the command streamer doesn't result in flushing/invalidating the IA Coherent lines from L3. However this can be achieved by setting the "**Pipe line flush Coherent lines**" control bit in the "L3SQCREG4" register.	
SLM comes up in an in-consistent state post reconfiguration and must be initialized by the driver for proper parity generation	CHV, BSW

DWord	Bit	Description
0	31:25	**All L3 Client Pool** Project: All Access: R/W Number of ways allocated for the all client pool. This is a combined pool for all clients. Value: 30h — Name: **[Default]** — Project: CHV, BSW **Programming Notes** When this field is non-zero, **DC Way Assignment** and **Read Only Client Pool** should be 0KB. Odd number values are not allowed. **Please refer to L3 Section with Allocation and Programming for recommended settings.**
	24:18	**DC Way Assignment** Project: All Access: R/W Number of ways allocated for DC. Note this allocation is only for DC data types. **Programming Notes** Note: This field must be 0KB if All L3 Client Pool is non-zero. Odd number values are not allowed. **Please refer to L3 HAS for valid programming values**

L3CNTLREG - L3 Control Register

17:11 **Read Only Client Pool**

Project:	All
Access:	R/W

Number of ways allocated for Read Only L3 clients. This is a combined pool for all Read Only clients.

Programming Notes
Note: This field must be 0KB if All L3 Client Pool is non-zero. Odd number values are not allowed. **Please refer to L3 HAS for valid programming values**

10 **Reserved**

Access:	R/W
Format:	PBC

9 **Error Detection Behavior Control**

Project:	CHV, BSW
Access:	R/W
Format:	Enable

The L3 error detection can be enabled to hang the GPU on a non-recoverable error due to SER type events. Such option will be used when corresponding context has data consistency requirements. Once error detection is enabled, s/w has to initialize URB or SLM to all 0's (based on usage model) prior to execution of the workload. Initialization is required to clean up the error detection logic and syndrome tracking.

Value	Name	Description	Project
0h	**[Default]**	RTL does not hang on parity errors or double bit error	CHV, BSW
1h		RTL enforces a hang on parity errors or double bit error	CHV, BSW

8 **GPGPU L3 Credit Mode Enable**

Project:	All
Access:	R/W
Format:	Enable

This bit is required to be enabled under GPGPU workloads to provide the MAX latency coverage from L3 cache. It will override the registers 0xB100[18:14] and 0xB100[23:19], to 0 and the maximum value respectively.

7:1 **URB Allocation**

Project:	All
Access:	R/W

Number of ways allocated for URB usage

Value	Name	Project
30h	**[Default]**	CHV, BSW

Programming Notes
Odd number values are not allowed. **Please refer to L3 HAS for valid programming values**

L3CNTLREG - L3 Control Register

	0	**SLM Mode Enable** Project: All Access: R/W Format: Enable When enabled, a 64KB (per bank) region of L3 is reserved for SLM.

L3 Control Register1

L3CNTLREG1 - L3 Control Register1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x8C47FF80 CHV, BSW
Size (in bits):	32
Address:	0B10Ch

DWord	Bit	Description
0	31:28	**Data Fifo Depth Control** Access: R/W Data Fifo Depth Control (TS mode). Value cannot be zero for normal operation. lbcf_csr_lc_datafifo_depth[3:0]. Value: 1000b — Name: **[Default]** — Project: CHV, BSW
	27:24	**Data Clock off time** Default Value: 1100b Access: R/W Data Clock off time (DATACLKOFF): Data Clock off time - Data block is shut off after these many number of clocks programmed in this register bits. lbcf_csr_lc_dataclkoff_time[3:0].Min value to be 4'h0100. It should be between 4'h4 : 4'hf.
	23:20	**TAG CLK OFF TIME** Default Value: 0100b Access: R/W
	19	**L3 Aging Disable Bit** Default Value: 0b Access: R/W L3 Aging Disable Bit (L3AGDIS): Aging Disable. lbcf_csr_lc_agingdis.
	18:15	**Fill aging** Default Value: 1111b

L3CNTLREG1 - L3 Control Register1

	Access:	R/W

Fill aging (L3AGF):
Aging Counter for Fill.
Ibcf_csr_lc_fill_aging_cnt[3:0].If bit B103.19 is 0 then this register value has to be nonzero.

14:11 **Aging Counter for Read 1 Port**

Default Value:	1111b
Access:	R/W

Aging Counter for Read 1 Port (L3AGR1):
Aging Counter for Read 1 Port.
Ibcf_csr_lc_r1_aging_cnt[3:0].If bit B103.19 is 0 then this register value has to be nonzero.

10:7 **L3 Aging Counter for R0**

Default Value:	1111b
Access:	R/W

L3 Aging Counter for R0 (L3AGR0):
Aging Counter for R0 Port.
Ibcf_csr_lc_r0_aging_cnt[3:0].If bit B103.19 is 0 then this register value has to be nonzero.

6:0 **Reserved**

Default Value:	000000b
Project:	CHV, BSW
Access:	RO

Reserved.

L3 LRA 0

L3_LRA_0 - L3 LRA 0		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x0E037C00	
Size (in bits):	32	
Exists If:	Device[Platform] == 'Client'	
Address:	04A10h	

DWord	Bit	Description
0	31:30	**L3** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min** Default Value: 0011100000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max** Default Value: 0011011111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 0 GPGPU

L3_LRA_0_GPGPU - L3 LRA 0 GPGPU

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0400FC00
Size (in bits):	32
Address:	04DD0h

DWord	Bit	Description
0	31:30	**L3 GPGPU** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min GPGPU** Default Value: 0001000000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max GPGPU** Default Value: 0000111111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min GPGPU** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 1

L3_LRA_1 - L3 LRA 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x67F701BF
Size (in bits):	32
Exists If:	Device[Platform] == 'Client'
Address:	04A14h

DWord	Bit	Description
0	31:30	**DC** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max** Default Value: 1001111111b Access: R/W Maximum value of programmable LRA2.
	19:10	**L3 LRA2 Min** Default Value: 0111000000b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max** Default Value: 0110111111b Access: R/W Maximum value of programmable LRA1.

L3 LRA 1 GPGPU

L3_LRA_1_GPGPU - L3 LRA 1 GPGPU

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x5FF6C1AF
Size (in bits):	32
Address:	04DD4h

DWord	Bit	Description
0	31:30	**DC GPGPU** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max GPGPU** Default Value: 0111111111b Access: R/W Maximum value of programmable LRA2.
	19:10	**L3 LRA2 Min GPGPU** Default Value: 0110110000b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max GPGPU** Default Value: 0110101111b Access: R/W Maximum value of programmable LRA1.

L3 LRA 2

L3_LRA_2 - L3 LRA 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000002
Size (in bits):	32
Exists If:	Device[Platform] == 'Client'
Address:	04A18h

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 LRA 2 GPGPU

L3_LRA_2_GPGPU - L3 LRA 2 GPGPU

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000002
Size (in bits):	32
Address:	04DD8h

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture GPGPU** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 LRA 0 3D

L3_LRA_0_3D - L3 LRA 0 3D	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0400FC00
Size (in bits):	32
Address:	04A10h

DWord	Bit	Description
0	31:30	**L3 3D** Default Value: 00b Access: R/W Which LRA should L3 use.
	29:20	**L3 LRA1 Min 3D** Default Value: 0001000000b Access: R/W Minimum value of programmable LRA1.
	19:10	**L3 LRA0 Max 3D** Default Value: 0000111111b Access: R/W Maximum value of programmable LRA0.
	9:0	**L3 LRA0 Min 3D** Default Value: 0000000000b Access: R/W Minimum value of programmable LRA0.

L3 LRA 1 3D

L3_LRA_1_3D - L3 LRA 1 3D

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x5FF2408F CHV, BSW
Size (in bits):	32
Address:	04A14h

DWord	Bit	Description
0	31:30	**DC 3D** Default Value: 01b Access: R/W Which LRA should DC use.
	29:20	**L3 LRA2 Max 3D** Access: R/W Maximum value of programmable LRA2. Value: 0111111111b — Name: **[Default]** — Project: CHV, BSW
	19:10	**L3 LRA2 Min 3D** Default Value: 0010010000b Access: R/W Minimum value of programmable LRA2.
	9:0	**L3 LRA1 Max 3D** Default Value: 0010001111b Access: R/W Maximum value of programmable LRA1.

L3 LRA 2 3D

L3_LRA_2_3D - L3 LRA 2 3D		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW, :GT2:B	
Source:	PRM	
Default Value:	0x00000002	
Size (in bits):	32	
Address:	04A18h	

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1:0	**Texture 3D** Default Value: 10b Access: R/W Which LRA should Texture use.

L3 Messaging Register

MSG_L3_LPFC - L3 Messaging Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08038h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:2	**Reserved** Access: RO
	1	**Acknowledge that L3 Unblock Completed** Access: R/W Acknowledge that L3 Unblock Completed 1'b0 : L3 unblock not complete yet (default) 1'b1 : L3 unblock has completed gpmunit self-clears this bit upon sampling.
	0	**Acknowledge that L3 Flush and Block Completed** Access: R/W Acknowledge that L3 Flush and Block Completed 1'b0 : L3 flush and block not complete yet (default) 1'b1 : L3 flush and block has completed gpmunit self-clears this bit upon sampling.

L3 SLM Register

L3SLMREG - L3 SLM Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x40000000
Size (in bits):	32
Address:	0B110h

DWord	Bit	Description
0	31	**Disable Periodic SLM/SQ slot allocation** Default Value: 0b Access: R/W Disable Periodic SLM/SQ slot allocation: When cfg_lslm_livelock_fairarb_dis=1 lslm unit always has the higher priority and lslm_lsqc_block to lsqcunit is asserted as long as there are requests in SLM FIFO. lbcf_csr_lslm_livelock_fairarb_dis.
	30:26	**LSLM_SQ_PENDING_MAX** Default Value: 10000b Access: R/W If lslmunit has read data to be sent to lcbrunit this cfg register specifies the maximum number of clocks for which LSLMunit can block SQ request from being sent o lcbrunit. Default value = 8. Value cannot be zero. lbcf_csr_lslm_sqpend_max[4:0].
	25	**LSLM address disable** Default Value: 0b Access: R/W **Description** 0 - Enable b2b addr matching fix. lslmunit should not block the cycle in fifo if there is a match in the pipeline. 1 - Disable b2b addr matching fix. lslmunit should block the cycle in fifo if there is a match in the pipeline. lbcf_csr_lslm_same_addr_dis. Default = 0. Set this bit to 1'b1 to workaround Atomic b2b bug on SLM for CHV, BSW A-step only.
	24:0	**Reserved** Access: RO

L3 SQC register 4

L3SQCREG4 - L3 SQC register 4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x40400000 CHV, BSW
Size (in bits):	32
Address:	0B118h

DWord	Bit	Description
0	31	**Reserved**
	30	**L3SQ URB Read CAM Match Disable** Default Value: 1b Access: R/W L3SQ URB Read CAM Match Disable (SQURBRDCAMDIS): Disables the L3SQ Cam Match ability for URB Reads. By disabling, this allows a performance mode where URB reads are not dependent upon one another but only on any previous URB writes to the same address. This allows many URB reads to the same cacheline at any given time instead of serializing the requests. 1 = URB Read CAM matching is disabled; multiple URB reads to the same cacheline are allowed to be concurrent (default). 0 = URB Read CAM matching is enabled; multiple URB reads to the same cacheline are serialized. lbcf_csr_lsqc_urbrdcam_dis.
	29:28	**Traffic regulation in LSQC for URB lookup traffic** Default Value: 00b Access: R/W Traffic regulation in LSQC for URB lookup traffic (URB lookups are issued to ltcc these many clocks apart). 00b - Continuous. 01b - 4 clocks apart. 10b - 8 clocks apart. 11b - 16 clocks apart. lbcf_lsqc_urb_traffic.
	27	**LQSC RO PERF DIS** Default Value: 0b Access: R/W Default: 0. when set, RO performance mode is disabled and all Reads proceed only after Parent recycles. lbcf_csr_lsqc_roperf_dis.

L3SQCREG4 - L3 SQC register 4

26	**Order Cam Snp Reject** Default Value: 0b Access: R/W Default: 0. when set, all slots resulting in matches to snp addr result in snprsp as REJECT instead of MISS. lbcf_csr_lsqc_ordercam_snpreject.
25	**LQSC RW PERF DIS** Default Value: 0b Access: R/W Default: 0. 0: Performance mode is enabled. when set, Rd to RW performance mode is disabled and all cycles proceed only after Parent recycles. lbcf_csr_lsqc_rwperf_dis.
24	**LSQC read rtrn local crdt pre-consume disable** Default Value: 0b Access: R/W 0 - Default, LSQD consumes the LNE local slicecredit when read return pending. 1 - LSQD consumes read rtrn credit in the clock it is ready to send read return data. lbcf_csr_lsqd_rdtrn_precrdt_dis.
23	**LSQC Mem Write sqcam HITM response disable** Default Value: 0b Access: R/W 0 - Default. 1 - This disables any Memory Write from cache with HitM tag response to respond for SQCAMs. lbcf_csr_lsqc_sqcam_l3tagrsphitm_dis.
22	**Non-IA coherent atomics enable** Default Value: 1b Access: R/W 0: Atomics in GTI. 1: Atomics in L3 (non-IA atomic) (default). lbcf_csr_lsqc_glblatmcs_l3. Value of this bit should be same as LNCF register bit 0xb008[0].Value of this bit should be same as LBCF register bit 0xb11c[8].

L3SQCREG4 - L3 SQC register 4

	21	**Pipe line flush Coherent lines** Default Value: 0b Project: CHV, BSW Access: R/W 1: Treat pipeline flush as invalidating even coherent lines along with non coherent lines . 0: Flush invalidates non coherent lines only. Ibcf_csr_lsqc_pipeflush_coh.
	20:0	**Reserved** Project: CHV, BSW Access: RO

L3 SQC registers 1

L3SQCREG1 - L3 SQC registers 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00810000 CHV, BSW
Size (in bits):	32
Address:	0B100h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO Reserved.
	23:19	**L3SQ General Priority Credit Initialization** Project: CHV, BSW Access: R/W **Description** L3SQ General Priority Credit Initialization (SQGPCI): Number of general and high priority credits that SQ presents to L3 Arbiter blocks This inherently also determines the depth of the SQ; reduce the number of credits and SQ uses fewer slots. Any value not listed here is considered Reserved. Gen priority credits is always greater than high priority credits. When this register is programmed through KMD, the crclk DOP clk gating should be disabled before the programming and be enabled ~100 clocks after the programming is done. Value # General Credits 00000b 0 00001b 2 00010b 4 00011b 6 00100b 8 00101b 10 00110b 12 00111b

L3SQCREG1 - L3 SQC registers 1

14
01000b
16
01001b
18
01010b
20
01011b
22
01100b
24 (default)
01101b
26
01110b
28
01111b
30
10000b
32
Other values are not possible.
Need to go up to 32 credits.
lbcf_csr_lsqc_gen_credit_init[4:0].

34
10010
36
10011
38
10100
40
Other values are not possible.
^M
need to go upto 40 credits

Value	Name	Project
10000b	**[Default]**	CHV, BSW

18:14 **L3SQ High Priority Credit Initialization**

Default Value:	00100b
Project:	CHV, BSW
Access:	R/W

L3SQ High Priority Credit Initialization (SQHPCI):
Number of general and high priority credits that SQ presents to L3 Arbiter blocks. This inherently also determines the depth of the SQ; reduce the number of credits and SQ uses fewer slots.
Any value not listed here is considered Reserved.
gen priority credits is always greater than high priority credits.

L3SQCREG1 - L3 SQC registers 1

		When this register is programmed through KMD, the crclk DOP clk gating should be disabled before the programming and be enabled ~100 clocks after the programming is done. Value # High Pri Credits 00000b 0 00001b 2 00010b 4 00011b 6 00100b 8 (default) 00101b 10 00110b 12 00111b 14 01000b 16 01001b 18 01010b 20 01011b 22 01100b 24 01101b 26 01110b 28 01111b 30 10000b 32 Other values are not possible. lbcf_csr_lsqc_hp_credit_init[4:0].lbcf_csr_lsqc_hp_credit_init[4:0] ++ lbcf_csr_lsqc_gen_credit_init[4:0] should always be less than or equal to 32.
	13:10	**Reserved** Access: RO Reserved.

L3SQCREG1 - L3 SQC registers 1

	9	**L3SQ Read Once Enable for Sampler Client** Access: R/W L3SQ Read Once Enable for Sampler Client (SQROE): Enables Read Once indications to L3 Cache from SQ. Once enabled, any reads from Sampler client (MT) are sent as Read Once. 0 = (default) Reads from Sampler clients issue Read to L3 Cache. 1 = Reads from Sampler clients issue Read Once to L3 Cache. lbcf_csr_sampler_readonce_en.
	8:6	**Reserved** Access: RO Reserved.
	5:3	**L3SQ Outstanding L3 Fills** Access: R/W L3SQ Outstanding L3 Fills (SQOUTSL3F): Identifies the number of L3 Fills that can be outstanding before SQ throttles the fill requests to L3 Cache. This is not an exact limit, but instead it is used as a threshold to throttling. Once the fill count is greater than or equal to the threshold, then no fills are issued until the fill responses are received to bring the outstanding count back below the threshold. 000b = (default) No limit. 001b = 1 fill. 010b = 2 fills. 011b = 4 fills. 100b = 8 fills. 101b = 16 fills. 11Xb = Reserved. lbcf_csr_lsqc_outs_fill[2:0].
	2:0	**L3SQ Outstanding L3 Lookups** Access: R/W L3SQ Outstanding L3 Lookups (SQOUTSL3L): Identifies the number of L3 lookups that can be outstanding before SQ throttles the lookup requests to L3 Cache. This is not an exact limit, but instead it is used as a threshold to throttling. once the lookup count is greater than or equal to the threshold, then no lookups are issued until the lookup responses are received to bring the outstanding count back below the threshold. 000b = (default) No limit. 001b = 1 lookup. 010b = 2 lookups. 011b = 4 lookups. 100b = 8 lookups. 101b = 16 lookups. 11Xb = Reserved. lbcf_csr_lsqc_outs_lookup[2:0].

L3 SQC registers 2

L3SQCREG2 - L3 SQC registers 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00004567
Size (in bits):	32
Address:	0B104h

DWord	Bit	Description
0	31:17	**Reserved** Access: RO Reserved.
	16	**L3SQ Priority Selection Disable** Access: R/W **Description** L3SQ Priority Selection Disable (SQPRIDIS): Enables the use of priority selection based on client ID decodes. If disabled, all cycles in SQ are treated as same priority. 0 = (default) Priority selection is enabled. 1 = Priority selection is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority_cnt_disable. Workaround: If this bit is set to 1 then the following bits in register B108-B10B should have identical values 29:28, 27:26, 25:24, 23:22, 21:20, 19:18, 17:16, 15:14, 13:12, 11:10, 9:8, 7:6, 5:4, 3:2, 1:0.
	15	**L3SQ Priority 3 Pool Count Disable** Access: R/W **Description** L3SQ Priority 3 Pool Count Disable (SQPRI3CNTDIS): When set, priority3 pool becomes unlimited. And priority3 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 3 pool count is enabled. 1 = Priority 3 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority3_cnt_disable.

L3SQCREG2 - L3 SQC registers 2

		Workaround: If this bit is set to 1 then the following bits in register B108-B10B should have identical values 29:28, 27:26, 25:24, 23:22, 21:20, 19:18, 17:16, 15:14, 13:12, 11:10, 9:8, 7:6, 5:4, 3:2, 1:0.
	14:12	**L3SQ Priority 3 Pool Counter** Default Value: 100b Access: R/W L3SQ Priority 3 Pool Counter (SQPRI3CNT): The count of cycles is selected from priority3 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests. lbcf_csr_priority3_cnt[2:0].
	11	**L3SQ Priority 2 Pool Count Disable** Access: R/W **Description** L3SQ Priority 2 Pool Count Disable (SQPRI2CNTDIS): When set, priority2 pool becomes unlimited. And priority2 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 2 pool count is enabled. 1 = Priority 2 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority2_cnt_disable. Workaround: If this bit is set to 1 then the following bits in register B108-B10B should have identical values 29:28, 27:26, 25:24, 23:22, 21:20, 19:18, 17:16, 15:14, 13:12, 11:10, 9:8, 7:6, 5:4, 3:2, 1:0.
	10:8	**L3SQ Priority 2 Pool Counter** Default Value: 101b Access: R/W L3SQ Priority 2 Pool Counter (SQPRI2CNT): The count of cycles is selected from priority2 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests.

L3SQCREG2 - L3 SQC registers 2

		lbcf_csr_priority2_cnt[2:0].
	7	**L3SQ Priority 1 Pool Count Disable** Access: R/W **Description** L3SQ Priority 1 Pool Count Disable (SQPRI1CNTDIS): When set, priority1 pool becomes unlimited. And priority1 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 1 pool count is enabled. 1 = Priority 1 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority1_cnt_disable. Workaround: If this bit is set to 1 then the following bits in register B108-B10B should have identical values 29:28, 27:26, 25:24, 23:22, 21:20, 19:18, 17:16, 15:14, 13:12, 11:10, 9:8, 7:6, 5:4, 3:2, 1:0.
	6:4	**L3SQ Priority 1 Pool Counter** Default Value: 110b Access: R/W L3SQ Priority 1 Pool Counter (SQPRI1CNT): The count of cycles is selected from priority1 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = 128 requests. lbcf_csr_priority1_cnt[2:0].
	3	**L3SQ Priority 0 Pool Count Disable** Access: R/W **Description** L3SQ Priority 0 Pool Count Disable (SQPRI0CNTDIS): When set, priority0 pool becomes unlimited. And priority0 pool count value should not be used in reset of the remaining counters. 0 = (default) Priority 0 pool count is enabled. 1 = Priority 0 pool count is disabled. Should not be set when RO perf mode is enabled (by default this is enabled which is bit[27] of B118-B11Bh). lbcf_csr_priority0_cnt_disable.

L3SQCREG2 - L3 SQC registers 2

		Workaround: If this bit is set to 1 then the following bits in register B108-B10B should have identical values 29:28, 27:26, 25:24, 23:22, 21:20, 19:18, 17:16, 15:14, 13:12, 11:10, 9:8, 7:6, 5:4, 3:2, 1:0.
	2:0	**L3SQ Priority 0 Pool Counter** Default Value: 111b Access: R/W L3SQ Priority 0 Pool Counter (SQPRI0CNT): The count of cycles is selected from priority0 pool before switching to other priority pools. Count is used as the power of 2. 000b = 1 request. 001b = 2 requests. 010b = 4 requests. 011b = 8 requests. ... 111b = (default) 128 requests. lbcf_csr_priority0_cnt[2:0].

L3 SQC registers 3

L3SQCREG3 - L3 SQC registers 3		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00001ABF	
Size (in bits):	32	
Address:	0B108h	

DWord	Bit	Description
0	31:30	**Reserved** Access: RO Reserved.
	29:28	**SOLunit Priority Value** Access: R/W SOLunit Priority Value (SQSOLPRIVAL): Identifies the priority value for all cycles that are initiated by SOLunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_sol_priority[1:0].
	27:26	**GSunit Priority Value** Access: R/W GSunit Priority Value (SQGSPRIVAL): Identifies the priority value for all cycles that are initiated by GSunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_gs_priority[1:0].
	25:24	**TEunit Priority Value** Access: R/W TEunit Priority Value (SQTEPRIVAL): Identifies the priority value for all cycles that are initiated by TEunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3.

L3SQCREG3 - L3 SQC registers 3

		Ibcf_csr_te_priority[1:0].
	23:22	**CLunit Priority Value** Access: R/W CLunit Priority Value (SQCLPRIVAL): Identifies the priority value for all cycles that are initiated by CLunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_cl_priority[1:0].
	21:20	**TSunit Priority Value** Access: R/W TSunit Priority Value (SQTSPRIVAL): Identifies the priority value for all cycles that are initiated by TSunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_ts_priority[1:0].
	19:18	**SFunit Priority Value** Access: R/W SFunit Priority Value (SQSFPRIVAL): Identifies the priority value for all cycles that are initiated by SFunit. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_sf_priority[1:0].
	17:16	**SVSM Priority Value** Access: R/W SVSM Priority Value (SQSVSMPRIVAL): Identifies the priority value for all cycles that are initiated by SVSM. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_svsm_priority[1:0].
	15:14	**SARB Priority Value** Access: R/W

L3SQCREG3 - L3 SQC registers 3

		SARB Priority Value (SQSARBPRIVAL): Identifies the priority value for all cycles that are initiated by State Arbiter (SARB). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0 (default). 01b = Priority 1. 10b = Priority 2. 11b = Priority 3. Ibcf_csr_sarb_priority[1:0].
	13:12	**SBE Priority Value** Default Value: 01b Access: R/W SBE Priority Value (SQSBEPRIVAL): Identifies the priority value for all cycles that are initiated by SBE. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1 (default). 10b = Priority 2. 11b = Priority 3. Ibcf_csr_sbe_priority[1:0].
	11:10	**IC$ Priority Value** Default Value: 10b Access: R/W IC$ Priority Value (SQICPRIVAL): Identifies the priority value for all cycles that are initiated by Instruction Cache (IC$). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. Ibcf_csr_ic_priority[1:0].
	9:8	**TDL Priority Value** Default Value: 10b Access: R/W TDL Priority Value (SQTDLPRIVAL): Identifies the priority value for all cycles that are initiated by TDL. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. Ibcf_csr_tdl_priority[1:0].
	7:6	**DCunit Priority Value** Default Value: 10b

L3SQCREG3 - L3 SQC registers 3

		Access: R/W DCunit Priority Value (SQDCPRIVAL): Identifies the priority value for all cycles that are initiated by DC. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2 (default). 11b = Priority 3. lbcf_csr_dc_priority[1:0].
	5:4	**DAPR Priority Value** Default Value: 11b Access: R/W DAPR Priority Value (SQDAPRPRIVAL): Identifies the priority value for all cycles that are initiated by DAPR. Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_dapr_priority[1:0].
	3:2	**MTunit Priority Value** Default Value: 11b Access: R/W MTunit Priority Value (SQMTPRIVAL): Identifies the priority value for all cycles that are initiated by Sampler (MT). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_mt_priority[1:0].
	1:0	**LSQCunit Priority Value** Default Value: 11b Access: R/W LSQCunit Priority Value (SQPRIVAL): Identifies the priority value for all cycles that are initiated by Super Queue (L3 Evictions). Priority is used in the L3 Super Queue (L3SQ). 00b = Priority 0. 01b = Priority 1. 10b = Priority 2. 11b = Priority 3 (default). lbcf_csr_lsqc_priority[1:0].

LBCF config save msg

LBCFCSR - LBCF config save msg		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B2FCh	
This register is not context saved and is written by PM unit.		
DWord	**Bit**	**Description**
0	31:10	**Reserved** Access: RO
	9:0	**Context save bit** Access: R/W Hardware Clear Bit[9]: Power Context Save Request. 0: Power context save is not being requested (default). 1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. Bits[8:0]: QWord Credits for Power Context Save Request. Minimum Credits = 1: Unit may send 1 QWord pair (enough for first LRI at least). Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and consumes one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

LBCF DPF Error log register 0

LBCFPM00 - LBCF DPF Error log register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B130h

Slice0 Bank 0 subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb0_error_addr1[9:0].
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. lbcf_sb0_valid_error1.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb0_error_addr0[9:0].
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. lbcf_sb0_valid_error0.

LBCF DPF Error log register 1

LBCFPM01 - LBCF DPF Error log register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B134h

Slice0 Bank 0 subbank1 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb1_error_addr1[9:0].
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. lbcf_sb1_error_addr0[9:0].
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb1_error_addr0[9:0].
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. lbcf_sb1_valid_error0.

LBCF DPF Error log register 2

LBCFPM02 - LBCF DPF Error log register 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B138h

Slice0 Bank0 Subbank 2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb2_error_addr1[9:0].
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. lbcf_sb2_valid_error1.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb2_error_addr0[9:0].
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. lbcf_sb2_valid_error0.

LBCF DPF Error log register 3

LBCFPM03 - LBCF DPF Error log register 3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B13Ch

Slice0 Bank0 subbank3 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb3_error_addr1[9:0].
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. lbcf_sb3_valid_error1.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. lbcf_sb3_error_addr0[9:0].
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. lbcf_sb3_valid_error0.

LBCF DPF Error log register 4

LBCFERRLOG01 - LBCF DPF Error log register 4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B140h

Slice 0 Bank 1 Subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 1 Error log register 00.

LBCF DPF Error log register 5

LBCFERRLOG02 - LBCF DPF Error log register 5		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B144h	
Slice 0 Bank 1 Subbank1 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. Slice 0 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid

LBCF DPF Error log register 6

LBCFERRLOG03 - LBCF DPF Error log register 6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B148h

Slice 0 Bank 1 subbank2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. Slice 0 Bank 1 Error log register 00.

LBCF DPF Error log register 7

LBCFERRLOG04 - LBCF DPF Error log register 7		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B14Ch	
Slice0 Bank1 subbank3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The Address located in field 31:21 is valid. Slice 0 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The address located in field 15:5 is valid. Slice 0 Bank 1 Error log register 00.

LBCF DPF Error log register 8

LBCFERRLOG05 - LBCF DPF Error log register 8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B150h

Slice 0 Bank 2 Subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. The number of rows varies between 4K vs. 8K/16K subbanks which requires 10 bits vs 11 bits respectively. This field contains the row# with the error. Slice 0 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 16:5 is valid and Slice 0 Bank 2 Error log register 00.
	16:5	**Row Number for Error0** Access: R/W Valid Error: The error located in field 16:5 is valid and Slice 0 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 16:5 is valid. Slice 0 Bank 2 Error log register 00.

LBCF DPF Error log register 9

LBCFERRLOG06 - LBCF DPF Error log register 9		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B154h	
Slice 0 Bank 2 subbank1 Error log register		
DWord	**Bit**	**Description**
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. The number of rows varies between 4K vs. 8K/16K subbanks which requires 10 bits vs 11 bits respectively. This field contains the row# with the error. Slice 0 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 10

LBCFERRLOG07 - LBCF DPF Error log register 10

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B158h

Slice 0 Bank 2 subbank2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. The number of rows varies between 4K vs. 8K/16K subbanks which requires 10 bits vs. 11 bits respectively. This field contains the row# with the error. Slice 0 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 2 Error log register 00.

LBCF DPF Error log register 11

LBCFERRLOG08 - LBCF DPF Error log register 11		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B15Ch	
slice0 Bank 2 subbank3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. The number of rows varies between 4K vs. 8K/16K subbanks which requires 10 bits vs. 11 bits respectively. This field contains the row# with the error. Slice 0 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 2 Error log register 00.

LBCF DPF Error log register 12

LBCFERRLOG09 - LBCF DPF Error log register 12

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B160h

Slice 0 Bank 3 subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 3 Error log register 00.

LBCF DPF Error log register 13

LBCFERRLOG10 - LBCF DPF Error log register 13		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B164h	
Slice 0 Bank 3 subbank1 Error log register 00		
DWord	**Bit**	**Description**
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 14

LBCFERRLOG11 - LBCF DPF Error log register 14

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B168h

Slice 0 Bank 3 subbank2 Error log register 00

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 3 Error log register 00.

LBCF DPF Error log register 15

LBCFERRLOG12 - LBCF DPF Error log register 15		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B16Ch	
Slice0 Bank 3 subbank3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 0 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 0 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 0 Bank 3 Error log register 00.

LBCF DPF Error log register 16

LBCFERRLOG13 - LBCF DPF Error log register 16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B170h

Slice 1 bank 0 Subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 1 Error log register 00.

LBCF DPF Error log register 17

LBCFERRLOG14 - LBCF DPF Error log register 17

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B174h

Slice 1 bank 0 subbank1 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 18

LBCFERRLOG15 - LBCF DPF Error log register 18

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B178h

Slice 1 bank 0 subbank2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 1 Error log register 00.

LBCF DPF Error log register 19

LBCFERRLOG16 - LBCF DPF Error log register 19		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B17Ch	
Slice1 bank 0 subbank3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 1 Error log register 00.

LBCF DPF Error log register 20

LBCFERRLOG17 - LBCF DPF Error log register 20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B180h

Slice 1 bank 1 subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 0 Error log register 00.

LBCF DPF Error log register 21

LBCFERRLOG18 - LBCF DPF Error log register 21	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B184h

Slice 1 bank 1 subbank1 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 22

LBCFERRLOG19 - LBCF DPF Error log register 22

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B188h

Slice 1 bank 1 subbank2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 0 Error log register 00.

LBCF DPF Error log register 23

LBCFERRLOG20 - LBCF DPF Error log register 23		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B18Ch	
slice1 bank1 subbbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 0 Error log register 00.

LBCF DPF Error log register 24

LBCFERRLOG21 - LBCF DPF Error log register 24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B190h

Slice 1 bank 2 subbank0 Error log register 00

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 2 Error log register 00.

LBCF DPF Error log register 25

LBCFERRLOG22 - LBCF DPF Error log register 25		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B194h	
Slice 1 bank 2 subbank 1 Error log register 00		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 26

LBCFERRLOG23 - LBCF DPF Error log register 26

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B198h

Slice 1 bank 2 subbank 2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 2 Error log register 00.

LBCF DPF Error log register 27

LBCFERRLOG24 - LBCF DPF Error log register 27		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B19Ch	
Slice 1 bank 2 subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 bank 2 Error log register 00.

LBCF DPF Error log register 28

LBCFERRLOG25 - LBCF DPF Error log register 28

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1A0h

Slice 1 Bank 3 subbank 0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 Bank 3 Error log register 00.

LBCF DPF Error log register 29

LBCFERRLOG26 - LBCF DPF Error log register 29		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1A4h	
Slice 1 Bank 3 subbank 1 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 30

LBCFERRLOG27 - LBCF DPF Error log register 30

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1A8h

Slice 1 Bank 3 subbank 2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 Bank 3 Error log register 00.

LBCF DPF Error log register 31

LBCFERRLOG28 - LBCF DPF Error log register 31		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1ACh	
slice1 Bank 3 subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 1 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 1 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 1 Bank 3 Error log register 00.

LBCF DPF Error log register 32

LBCFERRLOG29 - LBCF DPF Error log register 32

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1B0h

Slice 2 Bank 0 subbank0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 0 Error log register 00.

LBCF DPF Error log register 33

LBCFERRLOG30 - LBCF DPF Error log register 33		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1B4h	
Slice 2 Bank 0 subbank 1 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 34

LBCFERRLOG31 - LBCF DPF Error log register 34

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1B8h

Slice 2 Bank 0 Subbank 2 Error log register 00

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 0 Error log register 00.

LBCF DPF Error log register 35

LBCFERRLOG32 - LBCF DPF Error log register 35		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1BCh	
Slice2 Bank 0 Subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 0 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 0 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 0 Error log register 00.

LBCF DPF Error log register 36

LBCFERRLOG33 - LBCF DPF Error log register 36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1C0h

Slice 2 Bank 1 Subbank 0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 1 Error log register 00.

LBCF DPF Error log register 37

LBCFERRLOG34 - LBCF DPF Error log register 37		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1C4h	
Slice 2 Bank 1 Subbank 1 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 38

LBCFERRLOG35 - LBCF DPF Error log register 38

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1C8h

Slice 2 Bank 1 Subbank 2 Error log register 00

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 1 Error log register 00.

LBCF DPF Error log register 39

LBCFERRLOG36 - LBCF DPF Error log register 39		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1CCh	
Slice 2 Bank 1 subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 1 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 1 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 1 Error log register 00.

LBCF DPF Error log register 40

LBCFERRLOG37 - LBCF DPF Error log register 40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1D0h

Slice 2 Bank 2 subbank 0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 2 Error log register 00.

LBCF DPF Error log register 41

LBCFERRLOG38 - LBCF DPF Error log register 41

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1D4h

Slice 2 Bank 2 subbank 1 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 42

LBCFERRLOG39 - LBCF DPF Error log register 42

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1D8h

Slice 2 Bank 2 subbank 2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 2 Error log register 00.

LBCF DPF Error log register 43

LBCFERRLOG40 - LBCF DPF Error log register 43		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1DCh	
Slice 2 Bank 2 subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 2 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 2 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 2 Error log register 00.

LBCF DPF Error log register 44

LBCFERRLOG41 - LBCF DPF Error log register 44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1E0h

Slice 2 Bank 3 subbank 0 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. and Slice 2 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is validcorresponding logical 16KB group should bypass this row. Slice 2 Bank 3 Error log register 00.

LBCF DPF Error log register 45

LBCFERRLOG42 - LBCF DPF Error log register 45		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1E4h	
Slice 2 Bank 3 subbank 1 Error log register		
DWord	**Bit**	**Description**
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid.

LBCF DPF Error log register 46

LBCFERRLOG43 - LBCF DPF Error log register 46

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0B1E8h

Slice 2 Bank 3 subbank 2 Error log register

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 3 Error log register 00.

LBCF DPF Error log register 47

LBCFERRLOG44 - LBCF DPF Error log register 47		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B1ECh	
slice 2 Bank 3 subbank 3 Error log register		

DWord	Bit	Description
0	31:21	**Row Number for Error 1** Access: R/W Row Number for Error1: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	20:17	**Reserved** Access: RO
	16	**Valid Error 1** Access: R/W Valid Error: The error located in field 31:21 is valid. Slice 2 Bank 3 Error log register 00.
	15:5	**Row Number for Error0** Access: R/W Row Number for Error0: The physical row Addr where the parity error has been detected in SLM Bank. Slice 2 Bank 3 Error log register 00.
	4:1	**Reserved** Access: RO
	0	**Valid Error 0** Access: R/W Valid Error: The error located in field 15:5 is valid. Slice 2 Bank 3 Error log register 00.

LBS config bits

LBSREG - LBS config bits

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x46000000 CHV, BSW
Size (in bits):	32
Address:	0B124h

Config Bits for LBS unit

DWord	Bit	Description
0	31:27	**Retry timer for lookup into LSQC** Default Value: 01000b Access: R/W Time between receiving Reject Response from LSQC and doing a snoop lookup request again onto LSQCunit. 00000b: 0 clocks. 00001b: 1 clocks. 00010b: 2 clocks. ... 01000b: 8 clocks (default value). ... 11111b: 32 clocks. lbcf_retry_timer[4:0].
	26	**Recycle parent faster in R/W perf mode** Default Value: 1b Access: R/W Arc into recycle as soon as parent becomes eligible to be recycled. 0: Disabled (recycle possible only when parent is recycled). 1: Enabled (default). lbcf_csr_lsqc_rwperf_quickrec.
	25	**Perf mode for Writes to same address** Default Value: 1b Project: CHV, BSW Access: R/W Performance improvement for writes to same address in L3: 0 - Performance mode is not enabled. 1 - Performance mode is enabled (default). lbcf_csr_lsqc_erlyrec.
	24:0	**Reserved** Access: RO

LEAKAGECOUNTER

LEAKAGECOUNTER - LEAKAGECOUNTER		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	1300D4h	
Leakage counter readout		
DWord	**Bit**	**Description**
0	31:0	**Leakage_Counter** Default Value: 00000000h Access: RO This is a live read of the 32-bit leakage counter. Reading this register does not cause the counter to clear. Only the write-write protocol from Punit to Gunit will casue the leakage counter to clear.

LEAKAGECOUNTERCTL

LEAKAGECOUNTERCTL - LEAKAGECOUNTERCTL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300D0h

Leakage counter control.

DWord	Bit	Description
0	31	**leakage_lock** Default Value: 0b Access: R/W Lock This is the lock bit for the leakage counter registers. 13_00D0-13_00DC.
	30:1	**Reserved** Default Value: 00000000h Access: RO Reserved
	0	**leakage_count_en** Default Value: 0b Access: R/W Lock Enable the leakage counters. When zero, the counters will be held to zero. As this transitions to 0-1, counting is enabled. Like the EMON Energy Counters, the leakage counter will perform a "sum of weights" for the 16 events(/subwells) described in 13_00D8 and 13_00DC.

LEAKAGEWEIGHT1

LEAKAGEWEIGHT1 - LEAKAGEWEIGHT1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	1300D8h	
Leakage weights for wells types 1 to 4.		

DWord	Bit	Description
0	31:24	**leakage_vdve** Default Value: 00h Access: R/W Lock Leakage weight for 'vdve' well type. One of these wells are expected on CHV, BSW.
	23:16	**Reserved** Default Value: 00h Access: R/W Lock
	15:8	**Reserved** Default Value: 00h Access: R/W Lock
	7:0	**leakage_aon** Default Value: 00h Access: R/W Lock Leakage weight for 'aon' well type. One of these wells are expected on CHV, BSW.

LEAKAGEWEIGHT2

LEAKAGEWEIGHT2 - LEAKAGEWEIGHT2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300DCh

Leakage weights for wells types 5 to 8.

DWord	Bit	Description
0	31:24	**leakage_eupair** Default Value: 00h Access: R/W Lock Leakage weight for 'eupair' well type. Eight of these wells are expected on CHV, BSW.
	23:16	**leakage_ss** Default Value: 00h Access: R/W Lock Leakage weight for 'ss' well type. Two of these wells are expected on CHV, BSW.
	15:8	**leakage_l3** Default Value: 00h Access: R/W Lock Leakage weight for 'l3' well type. One of these wells are expected on CHV, BSW.
	7:0	**leakage_ffsc** Default Value: 00h Access: R/W Lock Leakage weight for 'ffsc' well type. One of these wells are expected on CHV, BSW.

LNCF config save msg

LNCFCSR - LNCF config save msg		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B0FCh	
This register is not context saved and is written by pm unit.		
DWord	**Bit**	**Description**
0	31:10	**Reserved** Access: RO
	9:0	**Context save bit** Access: R/W Hardware Clear Bit[9].Power Context Save Request 0: Power context save is not being requested (default). 1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. Bits[8:0].QWord Credits for Power Context Save Request. Minimum Credits = 1: Unit may send 1 QWord pair (enough for first LRI at least). Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

Load Indirect Base Vertex

3DPRIM_BASE_VERTEX - Load Indirect Base Vertex	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02440h-02443h
Valid Projects:	

DWord	Bit	Description
0	31:0	**Base Vertex** Format: S31 This register is used to store the Base Vertex of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Instance Count

3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02438h-0243Bh
Valid Projects:	

DWord	Bit	Description
0	31:0	**Instance Count** This register is used to store the Instance Count of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Start Instance

3DPRIM_START_INSTANCE - Load Indirect Start Instance	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0243Ch-0243Fh
Valid Projects:	

DWord	Bit	Description
0	31:0	**Start Vertex** Format: U32 This register is used to store the Start Instance of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Start Vertex

3DPRIM_START_VERTEX - Load Indirect Start Vertex	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02430h-02433h
Valid Projects:	

DWord	Bit	Description
0	31:0	**Start Vertex** Format: U32 This register is used to store the Start Vertex of the 3D_PRIMITIVE command when Load Indirect Enable is set.

Load Indirect Vertex Count

3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02434h-02437h
Valid Projects:	

DWord	Bit	Description
0	31:0	**Vertex Count** Format: U32 This register is used to store the Vertex Count of the 3D_PRIMITIVE command when Load Indirect Enable is set.

LOW 2X FREQUENCY THRESHOLD

LOW2XFREQTHRESH - LOW 2X FREQUENCY THRESHOLD

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x000000A0
Size (in bits):	32
Address:	101128h

New for CHV, BSW.

BIOS/Driver programs the Low2x frequency threshold.

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved
	15:0	**L3FREQREG** Default Value: 00A0h Access: R/W This is control signal needed from clock unit that can be set at 1 when 2X clock frequency is less than or equal to 1GHz. It needs to be at 0 when 2X clock frequency is > 1GHz. Value when in HPLL mode: low2xthresh=0x00B4. This sets low2xfreq high whenever cu2x frequency is less than or equal to 800, 800, 1000, 800, and 800 for CZ400, CZ320, CZ333, CZ266, and CZ200 respectively. Value when in GPLL mode: low2xthresh=0x00A0. This sets low2xfreq high whenever cu2x frequency is less than or equal to 1000, 800, 834, 889, and 800 for CZ400, CZ320, CZ333, CZ266, and CZ200 respectively.

LPFC control register

LPFCCNTL - LPFC control register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B01Ch	
LPFC control register.		
DWord	**Bit**	**Description**
0	31	**LPFC enable signal** Access: R/W LPPC event collection enable signal. Incf_lpfc_cnt_en.
	30:0	**Reserved** Project: CHV, BSW Access: RO Reserved.

LTCD Error Injection Register

LBCFERR - LTCD Error Injection Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	0B12Ch

LTCD Error Inject control bits in LBCF

DWord	Bit	Description
0	31:25	**Reserved** Project: CHV, BSW Access: RO
	24	**LTISEQSL parity error interupt** Default Value: 0b Project: CHV, BSW Access: R/W Parity error interrupt to LTISEQ Slice.
	23	**Bank hang on parity disable** Default Value: 0b Project: CHV, BSW Access: R/W
	22	**Parity Error Injection Enable** Project: CHV, BSW Access: R/W 0: Disable parity error injection. 1: Enable parity error injection. lbcf_parity_err_inject_en. Do not Enable this when ECC Error injection is enabled.
	21	**Double Bit Error injection** Project: CHV, BSW Access: R/W 0: Default No error injected. 1: Double bit error is injected. lbcf_ecc_2bit_err_inject. Single bit Error Injection and double bit error injection are mutually exclusive.

LBCFERR - LTCD Error Injection Register

		Do not Enable this when Parity Error injection is enabled.
	20	**Single Bit Error injection** Project: CHV, BSW Access: R/W 0: Default No error injected. 1: Single bit error is injected. lbcf_ecc_1bit_err_inject. Single bit Error Injection and double bit error injection are mutually exclusive. Do not Enable this when Parity Error injection is enabled.
	19	**ECC Error Injection Enable** Project: CHV, BSW Access: R/W 0: Disable ECC error injection. 1: Enable ECC error injection. lbcf_ecc_err_inject_en. Do not Enable this when Parity Error injection is enabled.
	18:4	**Row address for error injection** Project: CHV, BSW Access: R/W Row address for which error is injected. For ECC error injection, the same row address is overloaded to inject ECC error for that particular row address. SLM configuration. For Address: Rowaddress [16:7] is applicable for SLM banks and Parity injection is enabled and Rowaddress[18:7] is applicable for ECC (for both NonSLM/SLM banks). Sub bank is applicable only for Parity injection. Only Subbank 0, 1, 2, 3 are supported and are only related to SLM Banks (4KB). Total 12 bits address. Bits 18:7: Row address. Bits 6:4: Sub-Bank number. lbcf_parity_err_inject_rowaddr[9:0].lbcf_parity_err_inject_subank[2:0].16:7 - For SLM parity only. lbcf_parity_err_inject_rowaddr[9:0].lbcf_ecc_err_inject_rowaddr[11:0].For ECC error injection, the same row address is overloaded to inject ECC error for that particular row address.
	3:2	**Bank ID for error injection** Project: CHV, BSW Access: R/W 00b: Inject in Bank 0. 01b: Inject in Bank 1. 10b: Inject in Bank 2. 11b: Inject in Bank 3 (error injection not supported for SLM bank3).

LBCFERR - LTCD Error Injection Register

		lbcf_err_inject_bankid[1:0].
	1:0	**Slice ID for Error injection** Project: CHV, BSW Access: R/W 00b: Inject error in Slice 0. 01b: Inject error in Slice 1. 10b: Inject error in Slice 2. lbcf_err_inject_sliceid.

MA

MA - MA

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00094h

Message Address

DWord	Bit	Description
0	31:2	**ADDRESS** Default Value: 00000000h Access: R/W MA: Lower 32-bits of the system specified message address, always DW aligned. When GVD issues an MSI interrupt as a MEMWR on the SCL, the memory address corresponds to the value of this field.
	1:0	**RESERVED** Default Value: 00b Access: RO Reserved

Main Graphic Arbiter Error Report

ERROR - Main Graphic Arbiter Error Report	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040A0h

This register is used to report different error conditions. Error bits are writable.

DWord	Bit	Description
0	31	**Reserved Error Bits 31** Default Value: 0b Access: R/W Future Use.
	30	**Reserved Error Bits 30** Default Value: 0b Access: R/W Future Use.
	29	**Reserved Error Bits 29** Default Value: 0b Access: R/W Future Use.
	28	**Reserved Error Bits 28** Default Value: 0b Access: R/W Future Use.
	27	**Reserved Error Bits 27** Default Value: 0b Access: R/W Future Use.
	26	**Reserved Error Bits 26** Default Value: 0b Access: R/W

ERROR - Main Graphic Arbiter Error Report

		Future Use.
	25	**Reserved Error Bits 25** Default Value: 0b Access: R/W Future Use.
	24	**Reserved Error Bits 24** Default Value: 0b Access: R/W Future Use.
	23	**Reserved Error Bits 23** Default Value: 0b Access: R/W Future Use.
	22	**Reserved Error Bits 22** Default Value: 0b Access: R/W Future Use.
	21	**Reserved Error Bits 21** Default Value: 0b Access: R/W Future Use.
	20	**Reserved Error Bits 20** Default Value: 0b Access: R/W Future Use.
	19	**Reserved Error Bits 19** Default Value: 0b Access: R/W Future Use.
	18	**Reserved Error Bits 18** Default Value: 0b

ERROR - Main Graphic Arbiter Error Report

	Access:	R/W
	Future Use.	
17	**Reserved Error Bits 17**	
	Default Value:	0b
	Access:	R/W
	Future Use.	
16	**Reserved Error Bits 16**	
	Default Value:	0b
	Access:	R/W
	Future Use.	
15	**Reserved Error Bits 15**	
	Default Value:	0b
	Access:	R/W
	ctx_fault_ctxt_not_prsmt_err - The Present (P) field in the context-entry used to process the DMA request is Clear.	
14	**Reserved Error Bits 14**	
	Default Value:	0b
	Access:	R/W
	ctx_fault_root_not_prsmt_err - The present (UP/LP) field in the root-entry used to process the untranslated request with PASID is 0.	
13	**Reserved Error Bits 13**	
	Default Value:	0b
	Access:	R/W
	ctx_fault_pasid_not_prsnt_err - PASID Table entry to be used does not have the PRESENT flag set. This means the PASID entry is not valid.	
12	**Reserved Error Bits 12**	
	Default Value:	0b
	Access:	R/W
	ctx_fault_pasid_ovflw_err - PASID Table size in extended context entry defines the number of PASIDs that will be supported. If hardware receives a PASID number outside the supported boundary, report as an error.	
11	**Reserved Error Bits 11**	
	Default Value:	0b

ERROR - Main Graphic Arbiter Error Report

	Access:	R/W
	ctx_fault_pasid_dis_err - Submission of advanced context where the PASID field is not enabled in the extended context entry.	
10	**Reserved Error Bits 10**	
	Default Value:	0b
	Access:	R/W
	rstrm_fault_nowb_atomic_err - All page table accesses in advanced context with A/D bits are considered as atomic operations in WB space. However if the memory type for the page table accesses come out as anything but WB, that is an error.	
9	**Reserved**	
8	**Unloaded PD Error**	
	Default Value:	0b
	Access:	R/W
	The Cache Line containing a PD entry being accessed, was marked as invalid in the last PD load cycle.	
7	**Reserved Error Bits 7**	
	Default Value:	0b
	Access:	R/W
	Future Use.	
6	**Reserved**	
5	**Reserved**	
4	**Reserved**	
3	**Reserved**	
2	**Invalid Page Directory Entry Error**	
	Default Value:	0b
	Access:	R/W
	PD entry's valid bit is 0.	
1	**Reserved**	
0	**TLB Page Fault Error**	
	Default Value:	0b
	Access:	R/W
	A TLB Page's GTT translation generated a page fault (GTT entry not valid).	

Main Graphic Arbiter Error Report 2

ERROR_2 - Main Graphic Arbiter Error Report 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040A4h

DWord	Bit	Description
0	31:0	**Main Graphic Arbiter Error Report 2** Default Value: 00000000h Access: RO Bit [31:6] - Reserved. Bit [5:0] - tlbpend_reg_faultcnt[5:0].

Main Graphic Arbiter Error Report 3

ERROR_3 - Main Graphic Arbiter Error Report 3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	040A8h

This register is used to report different error conditions. Error bits are writable.

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO
	15	**Error3 Bits 15** Default Value: 0b Access: R/W Future Use.
	14	**Error3 Error Bits 14** Default Value: 0b Access: R/W Future Use.
	13	**Error3 Error Bits 13** Default Value: 0b Access: R/W Future Use.
	12	**Error3 Error Bits 12** Default Value: 0b Access: R/W Future Use.
	11	**Error3 Error Bits 11** Default Value: 0b Project: CHV, BSW Access: R/W invalid_varmtrr_overlap_memtype_rd.

ERROR_3 - Main Graphic Arbiter Error Report 3

	10	**Error3 Error Bits 10** Default Value: 0b Project: CHV, BSW Access: R/W invalid_varmtrr_overlap_memtype_wr.
	9	**Error3 Error Bits 9** Default Value: 0b Project: CHV, BSW Access: R/W invalid_default_memtype_value_rd.
	8	**Error3 Error Bits 8** Default Value: 0b Project: CHV, BSW Access: R/W invalid_default_memtype_value_wr.
	7	**Error3 Error Bits 7** Default Value: 0b Project: CHV, BSW Access: R/W invalid_varmtrr_memtype_value_rd.
	6	**Error3 Error Bits 6** Default Value: 0b Project: CHV, BSW Access: R/W invalid_varmtrr_memtype_value_wr.
	5	**Error3 Error Bits 5** Default Value: 0b Project: CHV, BSW Access: R/W invalid_fixedmtrr_memtype_value_rd.
	4	**Error3 Error Bits 4**

ERROR_3 - Main Graphic Arbiter Error Report 3

		Default Value: 0b Project: CHV, BSW Access: R/W invalid_fixedmtrr_memtype_value_wr.
	3	**Error3 Error Bits 3** Default Value: 0b Project: CHV, BSW Access: R/W gttc_internal_error.
	2	Reserved
	1	**Error3 Error Bits 1** Default Value: 0b Access: R/W tlbpend_internal_error.
	0	**Error3 Error Bits 0** Default Value: 0b Access: R/W reg_wrid_internal_error.

Main Graphic Arbiter Error Report Register

GFX_ARB_ERROR_RPT - Main Graphic Arbiter Error Report Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	040A0h

This register is used to report error conditions. Error bits are writable.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:9	**Reserved**
	8	**Unloaded PD Error** The Cache Line containing a PD entry being accessed was marked as invalid in the last PD load cycle.
	7	**Reserved**
	6	**Reserved**
	5	**Reserved**
	4	**Reserved**
	3	**Hardware Status Page Fault Error** HWSP's GTT translation generated a page fault (GTT entry not valid).
	2	**Invalid Page Directory entry error** PD entry's valid bit is 0.
	1	**Context Page Fault Error** A Context Page's GTT translation generated a page fault (GTT entry not valid).
	0	**TLB Page Fault Error** A TLB Page's GTT translation generated a page fault (GTT entry not valid).

MASTER_INT_CTL

MASTER_INT_CTL - MASTER_INT_CTL

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	44200h

This register has the master enable for graphics interrupts and gives an overview of what interrupts are pending.
An interrupt pending bit will read 1b while one or more interrupts of that category are set (IIR) and enabled (IER).
All Pending Interrupts are ORed together to generate the combined interrupt.
The combined interrupt is ANDed with the Master Interrupt enable to create the master enabled interrupt.
The master enabled interrupt goes to PCI device 2 interrupt processing.
The master interrupt enable must be set before any of these interrupts will propagate to PCI device 2 interrupt processing.

DWord	Bit	Description
0	31	**MASTER_INTERRUPT_ENABLE** Default Value: 0b Access: R/W THIS IS THE MASTER CONTROL FOR GRAPHICS INTERRUPT. this must be enabled for any of these interrupts to propagate to PCI dev2 interrupt processing. 0b - master interrupt disable, 1b - master interrupt enable
	30	**PCU_INTERRUPT_PENDING** Default Value: 0b Access: RO This field indicates if iterrups of this category is pending.
	29:8	**Reserved** Default Value: 00000b Access: RO Reserved
	7	**WDBOX_OACS_INTERRUPT_PENDING** Default Value: 0b Access: RO This field indicates if iterrups of this category is pending.
	6	**VEBOX_INTERRUPT_PENDING** Default Value: 0b

MASTER_INT_CTL - MASTER_INT_CTL

	Access:	RO
	This field indicates if iterrups of this category is pending.	
5	**Reserved**	
4	**GTPM_INTERRUPT_PENDING**	
	Default Value:	0b
	Access:	RO
	This field indicates if iterrups of this category is pending.	
3	**VCS2_INTERRUPT_PENDING**	
	Default Value:	0b
	Access:	RO
	This field indicates if iterrups of this category is pending.	
2	**VCS1_INTERRUPT_PENDING**	
	Default Value:	0b
	Access:	RO
	This field indicates if iterrups of this category is pending.	
1	**BLITTER_INTERRUPT_PENDING**	
	Default Value:	0b
	Access:	RO
	This field indicates if iterrups of this category is pending.	
0	**RENDER_INTERRUPT_PENDING**	
	Default Value:	0b
	Access:	RO
	This field indicates if iterrups of this category is pending.	

Master start timer

MASTIMER - Master start timer		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000001	
Size (in bits):	32	
Address:	0B438h	
DWord	**Bit**	**Description**
0	31:0	**Master start timer value** Default Value: 00000000000000000000000000000001b Access: R/W Master Start Timer (MSTSTTMR). lpconf_lpfc_master_start_timer [31:0].So many clocks are expired before starting the rest of the counters. Time to wait is 256 * value clocks. Value for this register cannot be 0.

Max Outstanding Pending TLB Requests 0

GFX_PEND_TLB_0 - Max Outstanding Pending TLB Requests 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04034h

DWord	Bit	Description
0	31	**TEX Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Texture Cache. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	30	**Reserved** Default Value: 0b Access: RO
	29:24	**TEX TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.
	23	**DC Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Instruction Cache. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	22	**Reserved** Default Value: 0b Access: RO
	21:16	**DC TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.

GFX_PEND_TLB_0 - Max Outstanding Pending TLB Requests 0

	15	**VF Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Vertex Fetch. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	14	**Reserved** Default Value: 0b Access: RO
	13:8	**VF TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.
	7	**VMC Limit Enable bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Video Motion Compensation. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	6	**Reserved** Default Value: 0b Access: RO
	5:0	**VMC TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.

Max Outstanding Pending TLB Requests 1

GFX_PEND_TLB_1 - Max Outstanding Pending TLB Requests 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04038h

DWord	Bit	Description
0	31	**SOL Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the SOL. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	30	**Reserved** Default Value: 0b Access: RO
	29:24	**SOL TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.
	23	**L3 Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the L3. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	22	**Reserved** Default Value: 0b Access: RO
	21:16	**L3 TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.

GFX_PEND_TLB_1 - Max Outstanding Pending TLB Requests 1

	15	**RCZ Limit Enable Bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Render Depth Cache. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	14	**Reserved** Default Value: 0b Access: RO
	13:8	**RCZ TLB Limit Count** Default Value: 000000b Access: R/W RCZ TLB Limit Count Project: All Format: U6 This is the MAX number of Allowed internal pending read requests which require a TLB read.
	7	**RCC Limit Enable bit** Default Value: 0b Access: R/W This bit is used to enable the pending TLB requests limitation function for the Render Color Cache. When set, the number of internal pending read requests which require a TLB read does not exceed the programmed counter value.
	6	**Reserved** Default Value: 0b Access: RO
	5:0	**RCC TLB Limit Count** Default Value: 000000b Access: R/W This is the MAX number of Allowed internal pending read requests which require a TLB read.

MAX Requests Allowed - GAM

GFX_MAX_REQ_COUNT - MAX Requests Allowed - GAM

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x43F20101
Size (in bits):	32
Address:	04AA4h

Programmable Request Count - GAM

DWord	Bit	Description
0	31:26	**GAP Writes Max Request Limit Count** Default Value: 010000b Access: R/W This is the MAX number of Allowed Write Requests Count - These counters keep track of the accepted write requests from all GAP clients (RCZ, HiZ, Stc, RCC, L3). Minimum count value must be = 1.
	25:20	**CVS Max Request Limit Count** Default Value: 111111b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	19	**Reserved** Default Value: 0b Access: RO
	18:13	**L3 Max Request Limit Count** Default Value: 010000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	12	**Reserved** Default Value: 0b Access: RO

GFX_MAX_REQ_COUNT - MAX Requests Allowed - GAM

	11:6	**Z Request Limit Count** Default Value: 000100b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	5:0	**RCC Request Limit Count** Default Value: 000001b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.

MAX Requests Allowed - MFX

MEDIA_MAX_REQ_COUNT - MAX Requests Allowed - MFX

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x10201020
Size (in bits):	32
Address:	04AA0h

Programmable Request Count - MFX

DWord	Bit	Description
0	31:24	**GFX Max Request Limit Count** Default Value: 00010000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	23:16	**MFX Max Request Limit Count** Default Value: 00100000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	15:14	**Reserved** Default Value: 00b Access: RO
	13:8	**VLF Max Request Limit Count** Default Value: 010000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.
	7:6	**Reserved** Default Value: 00b Access: RO

MEDIA_MAX_REQ_COUNT - MAX Requests Allowed - MFX

	5:0	**MFX Max Request Limit Count** Default Value: 100000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each client. Requests are counted, regardless of kind of cycle (Miss/Hit/Present). Minimum count value must be = 1.

MAX Requests Allowed - VEBX and BLT

VEBX_BLIT_MAX_REQ_COUNT - MAX Requests Allowed - VEBX and BLT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x08081020
Size (in bits):	32
Address:	04AA8h

Programmable Request Count - VEBX and BLT

DWord	Bit	Description
0	31:24	**BLT Max Request Limit Count** Default Value: 00001000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present).
	23:16	**VEBX Max Request Limit Count** Default Value: 00001000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present).
	15:8	**VLF1 Max Request Limit Count** Default Value: 00010000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present).
	7:0	**MFX1 Max Request Limit Count** Default Value: 00100000b Access: R/W This is the MAX number of Allowed Requests Count - These counters keep track of the accepted requests from each engine. Requests are counted, regardless of kind of cycle (Miss/Hit/Present).

MBC Control Register

MBCTL - MBC Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0907Ch

MBC Control Register

DWord	Bit	Description
0	31:17	**ECORSVD** Access: R/W ECO purposes Reserved
	16	**Fuse Write to VCR as Nonposted** Access: R/W 0 - Default - MBCunit will send the VCR (Fuse) writes as Posted Write cycles 1 - MBCunit will send the VCR (Fuse) writes as Non Posted Write cycles
	15:8	**RSVD** Access: RO RSVD
	7	**Disable Wait for G3d Outbound emtpy in MAE** Access: R/W 0 - Wait for SQempty for MAE update Flow 1 - MBC MAE update FSM will not wait for the SQempty to complete the FSM
	6:5	**RSVD** Access: RO RSVD
	4	**RSVD** Access: RO RSVD
	3	**RSVD** Access: RO RSVD

MBCTL - MBC Control Register

	2	**BME Update Enable** Access: R/W BME update Enable - 0 - Default Mae Update is not Enable. MBC will ignore all theBME updates from SA. 1- BME update is Enabled.
	1	**MAE Update Enable** Access: R/W MAE update Enable - 0 - Default Mae Update is not Enable. MBC will ignore all the MAE updates from SA. 1- MAE update is Enabled. MBC will respond to the MAE updates.
	0	**RSVD** Access: RO 0 = Use MBC to GAM path with 8B reads (default) 1 = Use MBC to Config Agent register interface with 4B reads

MBDSM

MBDSM - MBDSM

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1080C0h

Mirror of base of data stolen memory. Simply a read-only alias to the configuration registers.
Used by the MBC for boot context fetches.
This register contains the base address of Graphics Data Stolen DRAM memory. Note: This register is in device 0, 0xB0. Mirrored into device2, 0x5C.Graphics Stolen Memory is within DRAM space. The base of stolen memory will always be below 4G

DWord	Bit	Description
0	31:20	**BDSM** Default Value: 000h Access: RO BDSM: BASE_OF_Data_STOLEN_MEMORY. This register contains bits 31 to 20 of the base address of Data stolen DRAM memory. For certain GTLC generated accesses, this base register will be added to GTLC provided offset address, forming the full physical address for the PFI fabric. This is also used as a base for VGA paged accesses.
	19:1	**RESERVED** Default Value: 00000h Access: RO Reserved
	0	**BDSM_LOCK** Default Value: 0b Access: RO When set to 1b, this bit will lock all the bits in this register, including itself

MBGSM

MBGSM - MBGSM

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	108100h

Mirror of base of graphics stolen memory. Simply a read-only alias to the configuration registers.
Used by the MBC for boot context fetches.
Base of GTT table in Gfx Stolen Memory Note : This register is located in device 0, 0xB4. Mirrored into Device 2.
The GTT table is located within Graphics Stolen Memory in DRAM space.
The base of stolen memory will always be below 4G.

DWord	Bit	Description
0	31:20	**BGSM** Default Value: 000h Access: RO BGSM: Gfx Base of GTT Stolen Memory. This register contains bits 31 to 20 of the base address of GTT Table in stolen DRAM memory. BIOS determines base of GTT stolen memory by subtracting the GTT graphics stolen memory size (PCI offset 50 bits 9:8) from the Graphics Base of Data stolen(PCI offset 5C bits 31:20).
	19:1	**RESERVED** Default Value: 00000h Access: RO Reserved
	0	**BGSM_LOCK** Default Value: 0b Access: RO This bit will lock all writeable settings in this register including itself

MD

MD - MD

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00098h

Message Data

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved
	15:0	**DATA** Default Value: 0000h Access: R/W MD: This 16-bit field is programmed by system software. This forms the lower word of data for the MSI write transaction.

MEDFW_ACK

MEDFW_ACK - MEDFW_ACK	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300BCh

This register contains the per thread force wake acknowledge bits for the Media power well.

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved.
	15:0	**FWAKEMEDIAACK** Default Value: 0000h Access: RO Force Wake Media request bits. Driver must poll on the corresponding bit to confirm that the well has woken. For example, if 13_00B8[0] is written to a 1 (along with 13_00B8[16]='1'), then bit0 of this register indicates when the force wake request has been completed.

Media 1 TLB Control Register

M1TCR - Media 1 TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04264h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

Media 2 TLB Control Register

M2TCR - Media 2 TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04268h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

Media forcewake request

MEDFW_REQ - Media forcewake request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300B8h

This register contains per thread force wake request bits for the Render power Well. The upper 16 bits act as masks for the lower 16 bits. Bit 31 masks bit 15 and bit 16 masks bit 0.

1. Driver writes to GPM force wake request bit. (VV will have a Render(13_00B0(15:0)) and a Media (13_00B8(15:0)) bits.)
2. The GPM responds by writing (via PLINK) to 1300B4[15:0] or 1300BC[15:0] register.
3. Driver polls (1300B4[15:0] and/or 1300BC[15:0])status until 1... indicating that that well has completed wake sequence.

Since the registers are per thread, only the specific bit that was forced should be checked for status.

DWord	Bit	Description
0	31:16	**FWAKEMEDIAREQMSK** Default Value: 0000h Access: RO Mask bits for lower 16 bits to avoid a read modify/write. If '0', the corresponding bit in [15:0] is not changed. If '1', the corresponding bit in [15:0] is changed to the value in [15:0]
	15	**FWAKEMEDIAREQ15** Default Value: 0b Access: R/W Force Wake Media request 15.
	14	**FWAKEMEDIAREQ14** Default Value: 0b Access: R/W Force Wake Render request 14.
	13	**FWAKEMEDIAREQ13** Default Value: 0b Access: R/W

MEDFW_REQ - Media forcewake request

		Force Wake Render request 13.
	12	**FWAKEMEDIAREQ12** Default Value: 0b Access: R/W Force Wake Render request 12.
	11	**FWAKEMEDIAREQ11** Default Value: 0b Access: R/W Force Wake Render request 11.
	10	**FWAKEMEDIAREQ10** Default Value: 0b Access: R/W Force Wake Render request 10.
	9	**FWAKEMEDIAREQ9** Default Value: 0b Access: R/W Force Wake Render request 9.
	8	**FWAKEMEDIAREQ8** Default Value: 0b Access: R/W Force Wake Render request 8.
	7	**FWAKEMEDIAREQ7** Default Value: 0b Access: R/W Force Wake Render request 7.
	6	**FWAKEMEDIAREQ6** Default Value: 0b Access: R/W Force Wake Render request 6.
	5	**FWAKEMEDIAREQ5** Default Value: 0b

MEDFW_REQ - Media forcewake request

		Access: R/W Force Wake Render request 5.
	4	**FWAKEMEDIAREQ4** Default Value: 0b Access: R/W Force Wake Render request 4.
	3	**FWAKEMEDIAREQ3** Default Value: 0b Access: R/W Force Wake Render request 3.
	2	**FWAKEMEDIAREQ2** Default Value: 0b Access: R/W Force Wake Render request 2.
	1	**FWAKEMEDIAREQ1** Default Value: 0b Access: R/W Force Wake Render request 1.
	0	**FWAKEMEDIAREQ0** Default Value: 0b Access: R/W Force Wake Render request 0.

Media Power Meter Counter

MPMCNT - Media Power Meter Counter		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A270h-0A273h	

DWord	Bit	Description
0	31	**Media Power Meter Counter Overflow** Access: RO
	30:0	**Media PWRMTR Counter** Access: RO

Media Power Meter Counter No Clear

MPMCNTCLR - Media Power Meter Counter No Clear	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A288h-0A28Bh

Formerly cleared the count and the overflow bit, but now it is just a read-only value.

DWord	Bit	Description
0	31	**Media Power Meter Counter Overflow No Clear** Access: RO Formerly cleared the overflow bit, but now it is just a read-only value.
	30:0	**Media PWRMTR Counter No Clear** Access: RO Formerly cleared the count, but now is just a read-only value.

MEDIARC0COUNTER

MEDIARC0COUNTER - MEDIARC0COUNTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	13811Ch

Description	Project
This register contains the total RC0 residency (Media powered on and clocks running) time that Media was in since boot. SOXi Context Save/Restore : No The 40-bit HW counter will wrap around. The only clear condition is CZ reset. When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported. The units are CZ clock cycles. It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization. 0x13_8104[5] controls if this register should count or if it should be gated: 0= clear, 1=count This register will be cleared when A024[8] is set to 1	CHV, BSW
This register contains the total RC0 residency (Media powered on and clocks running) time that Media was in since boot. SOXi Context Save/Restore : No The 40-bit HW counter will wrap around. The only clear condition is CZ reset. When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported. The units are CZ clock cycles. It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization. 0x13_8104[5] controls if this register should count or if it should be gated: 0= clear, 1=count	CHV, BSW

DWord	Bit	Description
0	31:0	**MEDIARC0TIME** Default Value: 00000000h Access: RO Media RC0 Residency Counter.

MEDIARC1COUNTER

MEDIARC1COUNTER - MEDIARC1COUNTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138114h

This register contains the total RC1 residency (Media powered on and clock gated) time that Media was in since boot.
SOXi Context Save/Restore : No
The 40-bit HW counter will wrap around. The only clear condition is CZ reset.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[3] controls if this register should count or if it should be gated: 0= clear, 1=count

DWord	Bit	Description
0	31:0	**MEDIARC1TIME** Default Value: 00000000h Access: RO Media RC1 Residency Counter.

MEDIARC6COUNTER

MEDIARC6COUNTER - MEDIARC6COUNTER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	13810Ch

This register contains the total RC6 residency (Media power gated and clock gated) time that Media was in since boot. The counter will wrap around.
SOXi Context Save/Restore : No
The time is given in units of CZ clock cycles. The counter will reset on CZ reset going high.
This means that a warm reset will also reset this counter and it will also wrap around when it reaches max with no indication that an overflow occurred.
This register will freeze the count value (stop counting, but not reset) when pmu_gvd_renwakeack_nczfwoh=1.
This register will count whenever pmu_gvd_renwakeack_nczfwoh=0.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[1] controls if this register should count or if it should be gated: 0= clear, 1=count

DWord	Bit	Description
0	31:0	**MEDIARC6TIME** Default Value: 00000000h Access: RO Media Residency Counter.

MEMBOUNDCOUNTER

MEMBOUNDCOUNTER - MEMBOUNDCOUNTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138120h

SOXi Context Save/Restore : No
The 40-bit HW counter will wrap around. The only clear condition is CZ reset.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[6] controls if this register should count or if it should be gated: 0= clear, 1=count

DWord	Bit	Description
0	31:0	**MEMBOUNDTIME** Default Value: 00000000h Access: RO Render RC0 Residency Counter.

Message Register

MSGREG - Message Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001
Size (in bits):	32
Address:	040D4h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Reserved.
	15	**GO_PROTOCOL_GAM_REQUEST15** Default Value: 0b Access: R/W Reserved.
	14	**GO_PROTOCOL_GAM_REQUEST14** Default Value: 0b Access: R/W Reserved.
	13	**GO_PROTOCOL_GAM_REQUEST13** Default Value: 0b Access: R/W Reserved.
	12	**GO_PROTOCOL_GAM_REQUEST12** Default Value: 0b Access: R/W Reserved.
	11	**GO_PROTOCOL_GAM_REQUEST11** Default Value: 0b Access: R/W Reserved.
	10	**GO_PROTOCOL_GAM_REQUEST10** Default Value: 0b Access: R/W Reserved.

MSGREG - Message Register

	9	**GO_PROTOCOL_GAM_REQUEST9** Default Value: 0b Access: R/W Reserved.
	8	**GO_PROTOCOL_GAM_REQUEST8** Default Value: 0b Access: R/W Reserved.
	7	**GO_PROTOCOL_GAM_REQUEST7** Default Value: 0b Access: R/W Reserved.
	6	**GO_PROTOCOL_GAM_REQUEST6** Default Value: 0b Access: R/W Reserved.
	5	**GO_PROTOCOL_GAM_REQUEST5** Default Value: 0b Access: R/W Reserved.
	4	**GO_PROTOCOL_GAM_REQUEST4** Default Value: 0b Access: R/W Reserved.
	3	**GO_PROTOCOL_GAM_REQUEST3** Default Value: 0b Access: R/W Reserved.
	2	**GO_PROTOCOL_GAM_REQUEST2** Default Value: 0b Access: R/W Reserved.
	1	**GO_PROTOCOL_GAM_REQUEST1** Default Value: 0b Access: R/W Reserved.
	0	**GO_PROTOCOL_GAM_REQUEST0** Default Value: 1b

MSGREG - Message Register

		Access:	R/W
		0 - GPM to GAM Busy Ack Indication.	
		1 - GPM to GAM Idle Ack Indication.	

Messaging Register for GPMunit

MSG_GPM - Messaging Register for GPMunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x00000000
Size (in bits):	32
Address:	00C00h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
To set bit0, for example, the data would be 0x0001_0001.
To clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15	**GPM Messages Bit 15** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	14	**GPM Messages Bit 14** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	13	**GPM Messages Bit 13** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	12	**GPM Messages Bit 12** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	11	**GPM Messages Bit 11** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.

MSG_GPM - Messaging Register for GPMunit

	10	**GPM Messages Bit 10** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	9	**GPM Messages Bit 9** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	8	**GPM Messages Bit 8** Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	7	**GPM Messages Bit 7** Project: CHV, BSW Access: R/W Placeholder for GPM Messsages RPMunit could self-clear these bits upon sampling.
	6	**GPM Messages Bit 6** Project: CHV, BSW Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	5	**GPM Messages Bit 5** Project: CHV, BSW Access: R/W Placeholder for GPM Messsages. RPMunit could self-clear these bits upon sampling.
	4	**Request to send CPD Exit Ack Message on EventBus (U2C)** Access: R/W Request from GPMunit for RPMunit to send CPD_EXIT_ACK message on the Eventbus. RPMunit self-clears this bit upon sampling.
	3	**Request to send CPD Enter Ack Message on EventBus (U2C)** Access: R/W Request from GPMunit for RPMunit to send CPD_ENTER_ACK message on the Eventbus. RPMunit self-clears this bit upon sampling.

MSG_GPM - Messaging Register for GPMunit

	2	**Request to send Credit Active Deassert Message on EventBus (U2C)** Access: R/W Request from GPMunit for RPMunit to send CREDIT_ACTIVE_DEASSERT message on the Eventbus. RPMunit self-clears this bit upon sampling.
	1	**Request to send Credit Active Assert Message on EventBus (U2C)** Access: R/W Request from GPMunit for RPMunit to send CREDIT_ACTIVE_ASSERT message on the Eventbus. RPMunit self-clears this bit upon sampling.
	0	**Request to send IDI Shutdown Ack Message on EventBus (U2C)** Access: R/W Request from GPMunit for RPMunit to send IDI_SHUTDOWN_ACK message on the Eventbus. RPMunit self-clears this bit upon sampling.

Messaging Register for MDRBunit

MSG_MDRB - Messaging Register for MDRBunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x00000001
Size (in bits):	32
Address:	00C08h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
To set bit0, for example, the data would be 0x0001_0001.
To clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:2	**MDRB Messages** Access: R/W Placeholder for MDRB Messsages. MDRBunit could self-clear these bits upon sampling.
	1	**RFO Enable/Disable Ack for RPM (internal) RFO Request** Access: R/W RFO Enable/Disable Ack for Internal RFO Request. Enable Ack = 1'b1 Disable Ack = 1'b0
	0	**RFO Enable/Disable Ack for U2C (Evtentbus) RFO Request** Default Value: 1b Access: R/W RFO Enable/Disable Ack for U2C RFO Request. Enable Ack = 1'b1 Disable Ack = 1'b0

Messaging Register for MGSRunit

MSG_MGSR - Messaging Register for MGSRunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Default Value:	0x00000000
Size (in bits):	32
Address:	00C04h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
To set bit0, for example, the data would be 0x0001_0001.
To clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:0	**MGSR Messages** Access: R/W Placeholder for MGSR Messsages. MGSRunit could self-clear these bits upon sampling.

MFC_AVC_CABAC_INSERTION_COUNT

AVC_CABAC_INSERTION_COUNT - MFC_AVC_CABAC_INSERTION_COUNT	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128ACh
Valid Projects:	CHV, BSW

This register stores the count in bytes of **CABAC ZERO_WORD insertion. It is primarily provided for statistical data gathering.**

DWord	Bit	Description
0	31:0	**MFC AVC Cabac Insertion Count** Total number of bytes in the bitstream output before for the CABAC zero word insertion. This count is updated each time when the insertion count is incremented.

MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter

MFC_VIN_AVD_ERROR_CNTR - MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1

Address:	12804h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0 avd_error_flagsR[31:0]	31:0	**Reserved** Format: MBZ

MFC Image Status Control

MFC_IMAGE_STATUS_CONTROL - MFC Image Status Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128B8h
Valid Projects:	CHV, BSW

This register stores the suggested data for next frame in multi-pass.

DWord	Bit	Description
0	31:24	**Cumulative slice delta QP**
	23:16	**QP Value** suggested slice QP delta value for frame level Rate control. This value can be +ve or -ve
	15	**QP-Polarity Change** Cumulative slice delta QP polarity change.
	14:13	**Num-Pass Polarity Change** Number of passes after cumulative slice delta QP polarity changes.
	12	**Reserved** Project: CHV, BSW Format: MBZ
	11:8	**Total Num-Pass**
	7:4	**Reserved** Format: MBZ
	3	**Missing Huffman Code** Project: CHV, BSW Jpeg HW encoder reports if Huffman table entry is missing.
	2	**Panic** Panic triggered to avoid too big packed file.
	1	**Frame Bit Count** Frame Bit count over-run/under-run flag
	0	**Max Conformance Flag** Max Macroblock conformance flag or Frame Bit count over-run/under-run

MFC Image Status Mask

MFC_IMAGE_STATUS_MASK - MFC Image Status Mask	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128B4h
Valid Projects:	CHV, BSW

This register stores the image status(flags).

DWord	Bit	Description
0	31:0	**Control Mask** Control Mask for dynamic frame repeat.

MFC QP Status Count

MFC_QUP_CT - MFC QP Status Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128BCh
Valid Projects:	CHV, BSW

This register stores the suggested QP COUNTS in multi-pass.

DWord	Bit	Description
0	31:24	**Cumulative QP Adjust** Format: U8 Cumulative QP adjustment after multiple passes. If there is no need to multi-pass, this value would be zero. (This is in sign magnitude form).
	23:0	**Cumulative QP** Format: U24 Cumulative QP for all MB of a Frame (Can be used for computing average QP).

MFD Error Status

MFD_ERROR_STATUS - MFD Error Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12800h
Valid Projects:	CHV, BSW

This register stores the error status flags and count reports by the bit-stream decoder. This register is not part of hardware context save and restore. Driver should read the content prior to starting a new batch/frame.

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ This field is currently reserved
	19:16	**AVC Short Format Error Flags** Exists If: // AVC Short Format == Ture Bit-stream error detected by VLD short format bit-stream decoder**.** These flags are reset at the beginning of a frame and updated until starting of another frame. [19] – Slice Type SE Error Flag – Invalid Slice Type SE [18] – MMCO SE Error Flag – Invalid memory management control operation SE. MMCO Loop does not end (mmco control != 0) even after all MMCO SEs are decoded OR MMCO SEs are still being decoded and MMCO SE loop end (mmco control == 0) is hit [17] – Reordering IDC Error Flag – Syntax Element modification_of_pic_nums_idc >= 6 OR modification_of_pic_nums_idc != 3 (end of reordering loop) but reordering count has already hit maximum value [16] – Premature bitstream end is hit before finishing slice header decode
	15:0	**Bit-stream Error flags** Exists If: // AVC CAVLC, AVC CABAC, VC1 and MPEG2 == True Bitstream error detected by the VLD bit-steram decoder. These flags are reset at the beginning of a frame and updated until starting of another frame. AVC CAVLC: Please refer to AVC CAVLC table for each bit field AVC CABAC: Please refer to AVC CABAC table for each bit field VC1: Please refer to VC1 table for each bit field MPEG2: Please refer to MPEG2 table for each bit field

MFD Picture Parameter

MFD_PICTURE_PARAM - MFD Picture Parameter			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	VideoCS		
Default Value:	0x00000000		
Access:	R/W		
Size (in bits):	32		
Trusted Type:	1		
Address:	12820h		
Valid Projects:	CHV, BSW		
DWord	**Bit**	**Description**	
0	31:0	**Reserved**	
		Format:	MBZ

MFX_Memory_Latency_Count1

MFX_LAT_CT1 - MFX_Memory_Latency_Count1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12870h
Valid Projects:	CHV, BSW

This register stores the max and min memory latency counts reported on reference read requests. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the motion compensation engine is most likely hung waiting for read data to be returned from sub-system.
	15:8	**MFX Reference picture read request - Max Latency Count in 8xMedia clock cycles** This field reports the maximum memory latency count on all reference reads requested by the motion compensation engine.
	7:0	**MFX Reference picture read request - Min Latency Count in 8xMedia clock cycles** This field reports the minimum memory latency count on all reference reads requested by the motion compensation engine.

MFX0 Context Element Descriptor (High Part)

MFX0_CTX_EDR_H - MFX0 Context Element Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04444h

DWord	Bit	Description
0	31:0	**MFX0 Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W

MFX0 Context Element Descriptor (Low Part)

MFX0_CTX_EDR_L - MFX0 Context Element Descriptor (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04440h

DWord	Bit	Description
0	31:0	**MFX0 Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W

MFX0 Context Element Descriptor (Low Part)

MFX0_CTX_EDR_L - MFX0 Context Element Descriptor (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04440h

DWord	Bit	Description
0	31:0	**MFX0 Context Element Descriptor** Default Value: 00000009h Access: R/W

MFX0 PDP0/PML4/PASID Descriptor (High Part)

MFX0_CTX_PDP0_H - MFX0 PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0444Ch

DWord	Bit	Description
0	31:0	**MFX0 PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W

MFX0 PDP0/PML4/PASID Descriptor (Low Part)

MFX0_CTX_PDP0_L - MFX0 PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04448h

DWord	Bit	Description
0	31:0	**MFX0 PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W

MFX0 PDP1 Descriptor Register (High Part)

MFX0_CTX_PDP1_H - MFX0 PDP1 Descriptor Register (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04454h

DWord	Bit	Description
0	31:0	**MFX0 PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX0 PDP1 Descriptor Register (Low Part)

MFX0_CTX_PDP1_L - MFX0 PDP1 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04450h	

DWord	Bit	Description
0	31:0	**MFX0 PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX0 PDP2 Descriptor Register (High Part)

MFX0_CTX_PDP2_H - MFX0 PDP2 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0445Ch

DWord	Bit	Description
0	31:0	**MFX0 PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX0 PDP2 Descriptor Register (Low Part)

MFX0_CTX_PDP2_L - MFX0 PDP2 Descriptor Register (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04458h

DWord	Bit	Description
0	31:0	**MFX0 PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX0 PDP3 Descriptor Register (High Part)

MFX0_CTX_PDP3_H - MFX0 PDP3 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04464h

DWord	Bit	Description
0	31:0	**MFX0 PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX0 PDP3 Descriptor Register (Low Part)

MFX0_CTX_PDP3_L - MFX0 PDP3 Descriptor Register (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04460h

DWord	Bit	Description
0	31:0	**MFX0 PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX1 Context Element Descriptor (High Part)

MFX1_CTX_EDR_H - MFX1 Context Element Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04484h

DWord	Bit	Description
0	31:0	**MFX1 Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W

MFX1 Context Element Descriptor (Low Part)

MFX1_CTX_EDR_L - MFX1 Context Element Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	04480h

DWord	Bit	Description
0	31:0	**MFX1 Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W

MFX1 Context Element Descriptor (Low Part)

MFX1_CTX_EDR_L - MFX1 Context Element Descriptor (Low Part)			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000009		
Size (in bits):	32		
Address:	04480h		
DWord	**Bit**	**Description**	
0	31:0	**MFX1 Context Element Descriptor**	
		Default Value:	00000009h
		Access:	R/W

MFX1 PDP0/PML4/PASID Descriptor (High Part)

MFX1_CTX_PDP0_H - MFX1 PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0448Ch

DWord	Bit	Description
0	31:0	**MFX1 PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W

MFX1 PDP0/PML4/PASID Descriptor (Low Part)

MFX1_CTX_PDP0_L - MFX1 PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04488h

DWord	Bit	Description
0	31:0	**MFX1 PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W

MFX1 PDP1 Descriptor Register (High Part)

MFX1_CTX_PDP1_H - MFX1 PDP1 Descriptor Register (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04494h

DWord	Bit	Description
0	31:0	**MFX1 PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX1 PDP1 Descriptor Register (Low Part)

MFX1_CTX_PDP1_L - MFX1 PDP1 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04490h

DWord	Bit	Description
0	31:0	**MFX1 PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX1 PDP2 Descriptor Register (High Part)

MFX1_CTX_PDP2_H - MFX1 PDP2 Descriptor Register (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0449Ch

DWord	Bit	Description
0	31:0	**MFX1 PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX1 PDP2 Descriptor Register (Low Part)

MFX1_CTX_PDP2_L - MFX1 PDP2 Descriptor Register (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04498h

DWord	Bit	Description
0	31:0	**MFX1 PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX1 PDP3 Descriptor Register (High Part)

MFX1_CTX_PDP3_H - MFX1 PDP3 Descriptor Register (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044A4h

DWord	Bit	Description
0	31:0	**MFX1 PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

MFX1 PDP3 Descriptor Register (Low Part)

MFX1_CTX_PDP3_L - MFX1 PDP3 Descriptor Register (Low Part)		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	044A0h	

DWord	Bit	Description
0	31:0	**MFX1 PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

MFX Frame BitStream SE/BIN Count

MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1286Ch
Valid Projects:	CHV, BSW

This register stores the number of BINs (AVC CABAC) and SEs (CAVLD, VLD) decoded in a frame. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**MFX Frame Bit-stream SE/BIN Count** Total number of BINs/SEs decoded in current frame. This number is used with frame performance count to derive Bin/clk or SE/clk.

MFX Frame Macroblock Count

MFX_MB_COUNT - MFX Frame Macroblock Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12868h
Valid Projects:	CHV, BSW

This register stores the number of Macro-blocks decoded/encoded in current frame. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:20	**MBZ** Exists If: // JPEG == True Format: MBZ This field is currently reserved
	31:16	**Intra MB Count** Exists If: // AVC CAVLC, AVC CABAC, VC1 and MPEG2 == True Format: U16
	19:0	**JPEG Block Count** Exists If: // JPEG == True Format: U20 This 20-bit field indicates the number of 8x8 blocks within the JPEG frame. This field is clear at the start of decoding a new frame.
	15:0	**Number of MB Concealment** Exists If: // AVC CAVLC, AVC CABAC, VC1 and MPEG2 == True This 16-bit field indicates the number of MB is concealed by hardware. This field is clear at the start of decoding a new frame.

MFX Frame Motion Comp Miss Count

MFX_MISS_CT - MFX Frame Motion Comp Miss Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12888h
Valid Projects:	CHV, BSW

This register stores the total number of cacheline hits occurred in the motion compensation cache per frame. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**MFX Frame Motion Comp cache miss Count** Total number of CL misses occurred in the 12KB cache of the motion compensation engine per frame. This number is used along with **MFX Frame Motion Comp Read Count** to derive motion comp cache miss/hit ratio.

MFX Frame Motion Comp Read Count

MFX_READ_CT - MFX Frame Motion Comp Read Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12484h

This register stores the total number of reference picture read requests made by the Motion Compensation engine per frame. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ
	19:0	**MFX Frame Motion Comp CL read request Count** Total number of reference picture read requests by the motion compensation engine per frame.

MFX Frame Row-Stored/BitStream Read Count

MFX_ROW-PER-BS_COUNT - MFX Frame Row-Stored/BitStream Read Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12880h
Valid Projects:	CHV, BSW

This register stores the total number of row-stored/bit-stream read requests made by the pre-fetch engine per frame. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**MFX row-stored/bit-stream read request Count** Total number of row-stored/bit-stream read requests sent by the memory pre-fetch engine per frame.

MFX LRA 0

MFX_LRA_0 - MFX LRA 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x7F403F00
Size (in bits):	32
Address:	04A50h

DWord	Bit	Description
0	31:24	**MFX LRA1 Max** Default Value: 01111111b Access: R/W Maximum value of programmable LRA1.
	23:16	**MFX LRA1 Min** Default Value: 01000000b Access: R/W Minimum value of programmable LRA1.
	15:8	**MFX LRA0 Max** Default Value: 00111111b Access: R/W Maximum value of programmable LRA0.
	7:0	**MFX LRA0 Min** Default Value: 00000000b Access: R/W Minimum value of programmable LRA0.

MFX LRA 1

MFX_LRA_1 - MFX LRA 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0xFFC0BF80
Size (in bits):	32
Address:	04A54h

DWord	Bit	Description
0	31:24	**MFX LRA3 Max** Default Value: 11111111b Access: R/W Maximum value of programmable LRA3.
	23:16	**MFX LRA3 Min** Default Value: 11000000b Access: R/W Minimum value of programmable LRA3.
	15:8	**MFX LRA2 Max** Default Value: 10111111b Access: R/W Maximum value of programmable LRA2.
	7:0	**MFX LRA2 Min** Default Value: 10000000b Access: R/W Minimum value of programmable LRA2.

MFX LRA 2

MFX_LRA_2 - MFX LRA 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0000001B
Size (in bits):	32
Address:	04A58h

DWord	Bit	Description
0	31:8	**Reserved** Default Value: 000000h Access: RO
	7:6	**VCS LRA** Default Value: 00b Access: R/W Which LRA should VCS use.
	5:4	**VMX LRA** Default Value: 01b Access: R/W Which LRA should VMX use.
	3:2	**VMC LRA** Default Value: 10b Access: R/W Which LRA should VMC use.
	1:0	**VCR LRA** Default Value: 11b Access: R/W Which LRA should VCRSL1 use.

MFX LRA SL1 0

MFX_LRA_SL1_0 - MFX LRA SL1 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x7F403F00
Size (in bits):	32
Address:	04A60h

DWord	Bit	Description
0	31:24	**MFX SL1 LRA1 Max** Default Value: 01111111b Access: R/W Maximum value of programmable LRA1.
	23:16	**MFX SL1 LRA1 Min** Default Value: 01000000b Access: R/W Minimum value of programmable LRA1.
	15:8	**MFX SL1 LRA0 Max** Default Value: 00111111b Access: R/W Maximum value of programmable LRA0.
	7:0	**MFX SL1 LRA0 Min** Default Value: 00000000b Access: R/W Minimum value of programmable LRA0.

MFX LRA SL1 1

MFX_LRA_SL1_1 - MFX LRA SL1 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0xFFC0BF80
Size (in bits):	32
Address:	04A64h

DWord	Bit	Description
0	31:24	**MFX SL1 LRA3 Max** Default Value: 11111111b Access: R/W Maximum value of programmable LRA3.
	23:16	**MFX SL1 LRA3 Min** Default Value: 11000000b Access: R/W Minimum value of programmable LRA3.
	15:8	**MFX SL1 LRA2 Max** Default Value: 10111111b Access: R/W Maximum value of programmable LRA2.
	7:0	**MFX SL1 LRA2 Min** Default Value: 10000000b Access: R/W Minimum value of programmable LRA2.

MFX LRA SL1 2

MFX_LRA_SL1_2 - MFX LRA SL1 2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0000001B
Size (in bits):	32
Address:	04A68h

DWord	Bit	Description
0	31:8	**Reserved** Default Value: 000000h Access: RO
	7:6	**VCSSL1 LRA** Default Value: 00b Access: R/W Which LRA should VCSSL1 use.
	5:4	**VMXSL1 LRA** Default Value: 01b Access: R/W Which LRA should VMXSL1 use.
	3:2	**VMCSL1 LRA** Default Value: 10b Access: R/W Which LRA should VMCSL1 use.
	1:0	**VCRSL1 LRA** Default Value: 11b Access: R/W Which LRA should VCRSL1 use.

MFX Memory Latency Count2

MFX_LAT_CT2 - MFX Memory Latency Count2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12874h
Valid Projects:	CHV, BSW

This register stores the accumulative memory latency count on reference picture read requests. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:26	**Reserved** Format: MBZ
	25:0	**MFX Reference picture read request - Accumulative Memory Latency Count for the entire frame in 8xMedia clock cycles** The accumulative memory latency count of all reference reads requested by motion compensative engine per frame. This number is used with **MFX Frame Motion Comp Read Count** to derive average memory latency.

MFX Memory Latency Count3

MFX_LAT_CT3 - MFX Memory Latency Count3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12878h
Valid Projects:	CHV, BSW

This register stores the max and min memory latency counts reported on row-stored/bit-stream read requests. Max and current requests into memory sub-system engine. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:24	**Max Request Count** This field indicates the maximum number of requests allowed by the memory sub-system channel.
	23:16	**Current Request Count** This field indicates the number of requests currently outstanding in the memory sub-system. This field should report with a value of zero at the end of frame; otherwise the pre-fetch engine most likely hung waiting for read data to be returned from sub-system.
	15:8	**MFX row-stored/bit-stream read request - Max Latency Count in 8xMedia clock cycles** This field reports the maximum memory latency count on all row-stored/bit-stream reads requested by the memory pre-fetch engine.
	7:0	**MFX row-stored/bit-stream read request - Min Latency Count in 8xMedia clock cycles** This field reports the minimum memory latency count on all row-stored/bit-stream reads requested by the memory pre-fetch engine.

MFX Memory Latency Count4

MFX_LAT_CT4 - MFX Memory Latency Count4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1287Ch
Valid Projects:	CHV, BSW

This register stores the accumulative memory latency count on row-stored/bit-stream read requests. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:26	**Reserved** Format: MBZ
	25:0	**MFX row-stored/bit-stream read request - Accumulative Memory Latency Count for the entire frame in 8xMedia clock cycles** The accumulative memory latency count of all row-stored/bit-stream reads requested by pre-fetch engine per frame. This number is used with **Frame row-stored/bit-stream memory read count** to derive average memory latency.

MFX Pipeline Status Flags

MFX_STATUS_FLAGS - MFX Pipeline Status Flags

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12838h
Valid Projects:	CHV, BSW

This register stores the various pipeline status flags. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:17	**Reserved**
	16	**MFX Active** Frame decoding/encoding is in progress. Set on frame_start; clear on frame_end.
	15:10	**Reserved**
	9	**Streamout Enable**
	8	**Reserved**
	7	**Post Deblocking Mode Enable**
	6	**Pre Deblocking Mode Enable**
	5	**Decoder Mode Select** Value: 0 — Configure the MFD Engine for VLD Mode Value: 1 — Configure the MFD Engine for IT Mode
	4	**Codec Select** Value: 0 — Decode Value: 1 — Encode
	3:2	**Video Mode** Value: 00b — MPEG2 Value: 01b — VC1 Value: 10b — AVC Value: 11b — JPEG

MFX_STATUS_FLAGS - MFX Pipeline Status Flags

	1	**Decoder Short Format Mode**

Value	Name	Description
0		AVC/VC1 Short Format Mode is in use
1		AVC/VC1 Long Format Mode is in use

	0	**Stitch Mode**

Value	Name	Description
0b		Not in Stitch Mode
1b		In the Special Stitch Mode

MFX Slice Performance Count

MFX_SLICE_PERFORM_CT - MFX Slice Performance Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12864h
Valid Projects:	CHV, BSW

This register stores the number of clock cycles spent decoding/encoding the current slice. This register is not part of hardware context save and restore.

DWord	Bit	Description
0	31:0	**MFX Frame Performance Count** Total number of clocks between slice start and slice end. This count is incremented on crm_clk

MGGC

MGGC - MGGC

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000028
Size (in bits):	32
Address:	108040h

Mirror of GMCH Graphics Control Register. Simply a read-only alias to the configuration registers.
Used by the MBC for boot context fetches.

DWord	Bit	Description
0	31:15	**RESERVED** Default Value: 00000h Access: RO Reserved
	14	**VAMEN** Default Value: 0b Access: RO Enables the use of the iGFX engines for Versatile Acceleration. 1 - iGFX engines are in Versatile Acceleration Mode. Device 2 Class Code is 048000h. 0 - iGFX engines are in iGFX Mode. Device 2 Class Code is 030000h.
	13:10	**RESERVED** Default Value: 0h Access: RO
	9:8	**GGMS** Default Value: 00b Access: RO GTT Graphics Memory Size (GGMS): This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics Translation Table. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. GSM is assumed to be a contiguous physical DRAM space with DSM, and BIOS needs to allocate a contiguous memory chunk. Hardware will derive the base of GSM from DSM only using the GSM size programmed in the register. Hardware functionality in case of programming this value to Reserved is not guaranteed. 0h:No Preallocated Memory 1h: 2MB of Preallocated Memory 2h: 4MB of Preallocated Memory 3h: 8MB of Preallocated Memory

MGGC - MGGC

	7:3	**GMS** Default Value: 00101b Access: RO Graphics Mode Select(GMS): This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled. BIOS Requirement: BIOS must not set this field to 0h if IVD (bit 1 of this register) is 0. 0h = 0MB 1h = 32MB 2h = 64MB 3h = 96MB 4h = 128MB 5h = 160MB 6h = 192MB 7h = 224MB 8h = 256MB 9h = 288MB Ah = 320MB Bh = 352MB Ch = 384MB Dh = 416MB Eh = 448MB Fh = 480MB 10h = 512MB 11h = 8MB 12h = 12MB 13h = 16MB 14h = 20MB 15h = 24MB 16h = 28MB 17h = 36MB 18h = 40MB 19h = 44MB 1Ah = 48MB 1Bh = 52MB 1Ch = 56MB 1Dh = 60MB 1Eh = Reserved 1Fh = Reserved ..... 20h:1024MB (Not supported for CHV, BSW)

MGGC - MGGC

		..... 30h:1536MB (Not supported for CHV, BSW) ..... 40h:2048MB (Not supported for CHV, BSW) ..... 80h:4096MB (Not supported for CHV, BSW) 81h - FF:Reserved Other = Reserved When GMS != '0 (and VD=0): Address[31:0] is compared with VGA memory range. (The VGA memory range is A_0000h to B_FFFFh.). If there is a match and MSE = 1 and MEMRD or MEMWR, the access will route as a Rmdwvgamemen_cr cycle on the RMbus. If the RMbus returns a hit the GVD will select the command. As well, when 0 the GVD will check if scldown3_address[15:0] is one of the VGA IO register range. (The VGA IO range is 03B0h - 03BBh and 03C0h - 03DFh.) If there is a match and IOSE = 1 and the SCL command is either an IORD or IOWR, the GVD will intiate a (VGA) register cycle on the RMbus. If the RMbus returns a hit the GVD will select the command When GMS == '0 : No address compare will occur against VGA memory range or the VGA IO register range. Also, CC[15:8] is changed to 8'h80 from 8'h00
	2	**RESERVED** Default Value: 0b Access: RO Reserved
	1	**VGA_DISABLE** Default Value: 0b Access: RO VGA Disable (VD): 0: Enable. Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00. 1: Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub- Class Code field within Device 2 function 0 Class Code register is 80. BIOS Requirement: BIOS must not set this bit to 0 if the GMS field pre-allocates no memory. This bit MUST be set to 1 if Device 2 is disabled either via a fuse or fuse override (CAPID0[38] = 1) or via a register (DEVEN[3] = 0).
	0	**GGCLCK** Default Value: 0b Access: RO When set to 1b, this will lock all the bits in this register, including itself

MGSR2GAM Message Register

MGSR2GAM_MSGREG - MGSR2GAM Message Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	040D8h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO Mask Bits act as Write Enables for the bits[15:0] of this register.
	15	**MGSR2GAM Message Register 15** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	14	**MGSR2GAM Message Register 14** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	13	**MGSR2GAM Message Register 13** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	12	**MGSR2GAM Message Register 12** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	11	**MGSR2GAM Message Register 11** Default Value: 0b Access: R/W For Future Use.

MGSR2GAM_MSGREG - MGSR2GAM Message Register

		This bit is self clear.
	10	**MGSR2GAM Message Register 10** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	9	**MGSR2GAM Message Register 9** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	8	**MGSR2GAM Message Register 8** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	7	**MGSR2GAM Message Register 7** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	6	**MGSR2GAM Message Register 6** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	5	**MGSR2GAM Message Register 5** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	4	**MGSR2GAM Message Register 4** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	3	**MGSR2GAM Message Register 3** Default Value: 0b Access: R/W For Future Use. This bit is self clear.

MGSR2GAM_MSGREG - MGSR2GAM Message Register

	2	**MGSR2GAM Message Register 2** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	1	**MGSR2GAM Message Register 1** Default Value: 0b Access: R/W For Future Use. This bit is self clear.
	0	**MGSR2GAM Message Register 0** Default Value: 0b Access: R/W Bit0 - Tail Update Ack Message. This bit is self clear.

Misc. Clocking / Reset Control Registers

MISCCPCTL - Misc. Clocking / Reset Control Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000002
Size (in bits):	32
Address:	09424h

Miscellaneous Clocking / Reset Control Registers.

DWord	Bit	Description
0	31:8	**Bonus ECO bits** Access: R/W Bonus ECO bits
	7	**DOP clock gating enable for VEbox clks** Access: R/W Controls the Enabling of the DOP-level Vebox (cvclk) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled
	6	**DOP clock gating enable for Media clocks** Access: R/W Controls the Enabling of the DOP-level Media (cmclk) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled
	5	**Reserved** Access: RO Made Reserved field as we dont have Media 1 in CHV, BSW Controls the Enabling of the DOP-level Render (cmclk for 2nd media block) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled
	4	**Reserved**
	3	**DOP CLock gating Enable for Widi clocks** Access: R/W Controls the Enabling of the DOP-level Render (cwclk) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled

MISCCPCTL - Misc. Clocking / Reset Control Registers

	2	**DOP clock gating Enable for Fix clocks (cfclk)** Access: R/W Controls the Enabling of the DOP-level Render (cfclk/cf2xclk) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled
	1	**L1 Clock Ungate Enabling Control During Reset** Default Value: 1b Access: R/W Control to enable/disable L1 clock gating during soft resets and FLR reset processing 1 - disable L1 clock gating during soft resets and FLR 0 - enable L1 clock gating during soft resets and FLR (default op)
	0	**DOP Clock Gating Enable for Render Clocks** Access: R/W Controls the Enabling of the DOP-level Render (crclk/cr2xclk) Clock Gating via PM event messages 1 - Clock gating is enabled 0 - Clock gating is disabled

MISC. CTX control register

MISCCTXCTL - MISC. CTX control register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0942Ch	
DWord	**Bit**	**Description**
0	31:1	**Reserved** Access: RO Reserved.
	0	**Context Restore ACk indication from Csunit** Access: R/W Set Context Restore ACk indication from Csunit 1'b1 : Csunit has completed restoring CPunits address space Once set, CPunit hardware clears this bit after sending the ctx save ack done message to CS 1'b0 : Csunit has NOT completed restoring CPunits address space

Misc. Reset Control Register

RSTCTL - Misc. Reset Control Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	09420h	
Miscellaneous reset control registers.		

DWord	Bit	Description
0	31:4	**Reserved** Access: RO Reserved
	3:2	**Reset Staggering Period Control** Access: R/W Reset assertion staggering period between reset domains during FLR and soft-resets: 00: 24 cs clocks 01: 48 cs clocks 10: 72 cs clocks 11: 96 cs clocks
	1:0	**Reset Residency Control** Access: R/W Reset assertion residency period for FLR and soft-resets. 00: 24 cs clocks 01: 48 cs clocks 10: 96 cs clocks 11: 192 cs clocks

Miscellaneous Message Register for Power Managment Unit

MSG_MISC - Miscellaneous Message Register for Power Managment Unit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08048h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16].
In order to set bit0, for example, the data would be 0x0001_0001.;In order to clear bit0, for example, the data would be 0x0001_0000.
Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:3	**Reserved** Access: RO
	2	**Reserved**
	1	**Internal RFO Enable Ack (forwarded from MDRB to RPM to gpm)** Access: R/W From RPM on behalf of MGSR: Internal RFO Enable Acknowledgement b0 : RFO Disable Ack (default) b1 : RFO Enable Ack
	0	**GO Acknowledgement from OAunit** Access: R/W Go Acknowledgement b0 : Go=0 Ack <default> b1 : Go=1 Ack Normally the requirement is that when Go=0 request is sent, the agent has to indicate busy before sending the Go=0 acknowledgement. It can only go idle again once Go=1 is received. For OA, however, it does not have an idle indication to PM and thus do not have to do this.

MISR0

MISR0 - MISR0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182040h

Spare0 registers. Note: Changed for CHV, BSW.

DWord	Bit	Description
0	31:17	**Spare0** Default Value: 0000h Access: R/W Spare bits in case an ECO is needed.
	16	**Standby Threshold Periodic Enable** Default Value: 0000h Access: R/W Standby Threshold Periodic Enable
	15:0	**Standby Threshold** Default Value: 0000h Access: R/W

Description	Project
Standby Threshold For 33MHz cz freq (200MHz, 267MHz, 320MHz, 333MHz, 400MHz, 467MHz, 533MHz) 0 = 0 us 1 = 15.36 us 2 = 30.72 us 3 = 46.08 us FFFF ~= 1s For non-33MHz cz freq (350MHz, 356MHz, 360MHz, 373MHz), standby threshold will be in increments of (15.36us + (1-7)%)	CHV, BSW

MISR1

MISR1 - MISR1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182044h

Spare1 registers. Note: Changed for CHV, BSW.

DWord	Bit	Description
0	31:0	**Spare1** Default Value: 00000000h Access: R/W Spare bits in case an ECO is needed.

MISR2

MISR2 - MISR2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182048h

Spare2 registers. Note: Changed for CHV, BSW.

DWord	Bit	Description
0	31:0	**Spare2** Default Value: 00000000h Access: R/W Spare bits in case an ECO is needed.

MISR3

MISR3 - MISR3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	18204Ch

Spare3 registers. Note: Changed for CHV, BSW.

DWord	Bit	Description
0	31:0	**Spare3** Default Value: 00000000h Access: R/W Spare bits in case an ECO is needed.

MISR4

MISR4 - MISR4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182050h

Spare4 registers. Note: Changed for CHV, BSW.

DWord	Bit	Description
0	31:0	**Spare4** Default Value: 00000000h Access: R/W Spare bits in case an ECO is needed.

MMIO_INDEX

MMIO_INDEX - MMIO_INDEX

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00000h

Contains address and target.
Punit cannot access IO space from message channel.
A 32 bit IO write to this port loads the offset of the MMIO register or offsetinto the GTT that needs to be accessed.
An IO Read returns the current value of this register. An 8/16 bit IO write to this register is completed but does not update this register.
This mechanism to access internal graphics MMIO registers must not be used to access VGA IO registers which are mapped through the MMIO space. VGA registers must be accessed directly through the dedicated VGA IO ports. This is used by SBIOS. It is not used by graphics driver.
This register is only accessible through the IOSF Primary bus. The base register is defined by IOBAR.

DWord	Bit	Description
0	31:2	**Register_offset** Default Value: 00000000h Access: R/W This field selects GTT entry or any one of the Dword registers within the MMIO register space of this device.
	1:0	**Target** Default Value: 00b Access: R/W 00 = MMIO Registers, 01 = GTT, 1X =Reserved

Mode Register for GAB

GAB_MODE - Mode Register for GAB

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Address:	220A0h-220A3h

The GAB_MODE register contains information that controls configurations in the GAB.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask
	15:6	**Reserved** Read/Write
	5:3	**BLB Arbitration Priority** Format: U3
	2:0	**BCS Arbitration Priority** Format: U3

Mode Register for GAC

GAC_MODE - Mode Register for GAC

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000 CHV, BSW
Access:	r/w
Size (in bits):	32

Address:	120A0h-120A3h
ShortName:	GAC_MODE
Valid Projects:	CHV, BSW

The GAC_MODE register contains information that controls configurations in the GAC.

DWord	Bit	Description
0	31:16	**Mask** Access: WO Format: Mask
	15:1	**Reserved** Access: r/w
	0	**Reserved** Project: CHV, BSW Access: r/w

Mode Register for GAFS

GAFS_MODE - Mode Register for GAFS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	r/w
Size (in bits):	32
Trusted Type:	1
Address:	0212Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Access: WO Format: Mask Masks: These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15:11	**Reserved** Format: MBZ
	10	**Reserved** Project: CHV, BSW Format: MBZ
	9	**Reserved**
	8:2	**Reserved** Format: MBZ
	1:0	**Reserved** Project: CHV, BSW Format: MBZ

MSAC

MSAC - MSAC

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00010000
Size (in bits):	32
Address:	00060h

This register determines the size of the graphics memory aperture. Only the system BIOS will write this register based on pre-boot address allocation efforts. Graphics may read this register to determine the correct aperture size. System BIOS needs to save this value on boot so that it can reset it correctly during S3 resume. The size of the aperture must not be modified by software after its location is written into GMADR (offset 18h).

DWord	Bit	Description
0	31:21	**RESERVED** Default Value: 000h Access: RO Reserved
	20	**APSZ_4** Default Value: 0b Access: R/W This field is used to determine the size of Aperture (GMADR) and controls affects certain bits of GMADR register. 00000 = 128MB =>GMADR.B [26:4] is hardwired to 0 00001 = 256MB =>GMADR.B[27] = 0, RO 00010 = illegal (hardware will treat this as 00011) 00011 = 512MB =>GMADR.B[28:27] = 0, RO 00100-00110 = illegal (hardware will treat this as 00111) 00111 = 1024MB =>GMADR.B[29:27] = 0, RO 01000-01110= illegal (hardware will treat this as 01111) 01111= 2048MB=> GMADR.B[30:27] = 0, RO 10000-11110 = illegal (hardware will treat this as 11111) 11111= 4096MB => GMADR.B[31:27] = 0, RO
	19	**APSZ_3** Default Value: 0b Access: R/W This field is used to determine the size of Aperture (GMADR) and controls affects certain bits of GMADR register. 00000 = 128MB =>GMADR.B [26:4] is hardwired to 0 00001 = 256MB =>GMADR.B[27] = 0, RO 00010 = illegal (hardware will treat this as 00011)

MSAC - MSAC

		00011 = 512MB =>GMADR.B[28:27] = 0, RO 00100-00110 = illegal (hardware will treat this as 00111) 00111 = 1024MB =>GMADR.B[29:27] = 0, RO 01000-01110= illegal (hardware will treat this as 01111) 01111= 2048MB=> GMADR.B[30:27] = 0, RO 10000-11110 = illegal (hardware will treat this as 11111) 11111= 4096MB => GMADR.B[31:27] = 0, RO
	18	**APSZ_2** Default Value: 0b Access: R/W This field is used to determine the size of Aperture (GMADR) and controls affects certain bits of GMADR register. 00000 = 128MB =>GMADR.B [26:4] is hardwired to 0 00001 = 256MB =>GMADR.B[27] = 0, RO 00010 = illegal (hardware will treat this as 00011) 00011 = 512MB =>GMADR.B[28:27] = 0, RO 00100-00110 = illegal (hardware will treat this as 00111) 00111 = 1024MB =>GMADR.B[29:27] = 0, RO 01000-01110= illegal (hardware will treat this as 01111) 01111= 2048MB=> GMADR.B[30:27] = 0, RO 10000-11110 = illegal (hardware will treat this as 11111) 11111= 4096MB => GMADR.B[31:27] = 0, RO
	17	**APSZ_1** Default Value: 0b Access: R/W This field is used to determine the size of Aperture (GMADR) and controls affects certain bits of GMADR register. 00000 = 128MB =>GMADR.B [26:4] is hardwired to 0 00001 = 256MB =>GMADR.B[27] = 0, RO 00010 = illegal (hardware will treat this as 00011) 00011 = 512MB =>GMADR.B[28:27] = 0, RO 00100-00110 = illegal (hardware will treat this as 00111) 00111 = 1024MB =>GMADR.B[29:27] = 0, RO 01000-01110= illegal (hardware will treat this as 01111) 01111= 2048MB=> GMADR.B[30:27] = 0, RO 10000-11110 = illegal (hardware will treat this as 11111) 11111= 4096MB => GMADR.B[31:27] = 0, RO
	16	**APSZ_0** Default Value: 1b Access: R/W

MSAC - MSAC

		This field is used to determine the size of Aperture (GMADR) and controls affects certain bits of GMADR register. 00000 = 128MB =>GMADR.B [26:4] is hardwired to 0 00001 = 256MB =>GMADR.B[27] = 0, RO 00010 = illegal (hardware will treat this as 00011) 00011 = 512MB =>GMADR.B[28:27] = 0, RO 00100-00110 = illegal (hardware will treat this as 00111) 00111 = 1024MB =>GMADR.B[29:27] = 0, RO 01000-01110= illegal (hardware will treat this as 01111) 01111= 2048MB=> GMADR.B[30:27] = 0, RO 10000-11110 = illegal (hardware will treat this as 11111) 11111= 4096MB => GMADR.B[31:27] = 0, RO
	15:0	**RESERVED** Default Value: 0000h Access: RO Reserved

MSI_CAPID_MC

MSI_CAPID_MC - MSI_CAPID_MC

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0000B005
Size (in bits):	32
Address:	00090h

90-91h: Message Signaled Interrupts Capability ID
92-93h: Message Signaled Control Register

DWord	Bit	Description
0	31:24	**RESERVED** Default Value: 00h Access: RO Reserved
	23	**MODE_64B_ADDRCAP** Default Value: 0b Access: RO C64: 32-bit capable only
	22:20	**MULTIPLE_MESSAGE_ENABLE** Default Value: 000b Access: R/W MME: This field is RW for software compatibility, but only a single message is ever generated. System software programs this field to indicate the actual number of messages allocated to this device. This number will be equal to or less than the number actually requested. The encoding is the same as for the MMC field below.
	19:17	**MULTIPLE_MESSAGE_CAPABLE** Default Value: 000b Access: RO MMC: This device is only single message capable. System Software reads this field to determine the number of messages being requested by this device. Value: Number of requests 000: 1 001-111: Reserved

MSI_CAPID_MC - MSI_CAPID_MC

	16	**MSI_ENABLE** Default Value: 0b Access: R/W MSIE: If set, MSI is enabled and traditional interrupts are not used to generate interrupts. PCICMDSTS.BME must be set for an MSI to be generated. 0 : MSI interrupts are disabled. 1 : MSI interrupts are enabled. Permits sending an MSI interrupt.
	15:8	**POINTER_TO_NEXT_CAPABILITY** Default Value: B0h Access: R/W Once Points to the next item in the list(B0=VCID support). This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset.
	7:0	**CAPABILITY_ID** Default Value: 05h Access: RO CAPID: Indicates an MSI capability

MTRR Capability Register 0

MTRR_CR_0 - MTRR Capability Register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x0000050A
Size (in bits):	32
Address:	0F100h

Register to define MTRR - range register capabilities.

DWord	Bit	Description
0	31:11	**Reserved** Default Value: 00000000000000000000b Access: RO
	10	**Write Combining Support** Default Value: 1b Access: RO 0: Write Combining (WC) memory type is not supported. 1: Write Combining (WC) memory type is supported. GFX Implementation: More details on memory type section however WC support in GFX looks like streamlining non-cacheable accesses. This is the existing UC concept used in GFX architecture.
	9	**Reserved** Default Value: 0b Access: RO
	8	**Fixed Range MTRRs Support** Default Value: 1b Access: RO 0: No Fixed range MTRRs are supported. 1: Fixed Range MTRRs (IA32_MTRR_FIX64K_00000 through IA32_MTRR_FIX4K_0F8000) are supported.
	7:0	**Variable Range MTRR Count** Default Value: 0Ah Access: RO Indicates the number of variable ranges implemented.

MTRR Capability Register 1

MTRR_CR_1 - MTRR Capability Register 1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0F104h	
Register to define MTRR - range register capabilities		
DWord	**Bit**	**Description**
0	31:0	**MTRR Capability Register 1 Reserved** Default Value: 00000000h Access: RO Bit[63:32]: Reserved.

MTRR Default Type Register 0

MTRR_DT_0 - MTRR Default Type Register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F108h

Register to define MTRR - range register capabilities.

DWord	Bit	Description
0	31:12	**Reserved** Default Value: 00000000000000000000b Access: RO
	11	**Reserved**
	10	**Fixed Range MTRR Enable/Disable** Default Value: 0b Access: R/W 0: Disable fixed-range MTRRs. 1: Enable fixed-range MTRRs. When the fixed-range MTRRs are enabled, they take priority over the variable-range MTRRs when overlaps in ranges occur. If the fixed-range MTRRs are disabled, the variable range MTRRs can still be used and can map the range ordinarily covered by the fixed-range MTRRs. GFX Implementation: GFX uses this field as a specific enable/disable for fixed range MTRRs.
	9:8	**Reserved** Default Value: 00b Access: RO
	7:0	**Default Memory Type** Default Value: 00h Access: R/W Indicates default memory type used for physical memory address ranges that do not have a memory type specified for them by an MTRR. Legal values for this field are 0, 1, 4, 5, and 6. GFX Implementation: GFX uses this field to assign memory regions that are not assigned as part of the fixed and variable range registers.

MTRR Default Type Register 1

MTRR_DT_1 - MTRR Default Type Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F10Ch

Register to define MTRR - range register capabilities.

DWord	Bit	Description
0	31:0	**MTRR Default Type Register 1 Reserved** Default Value: 00000000h Access: RO Bit[63:32]: Reserved.

MT Virtual Page Address Registers

MTTLB_VA - MT Virtual Page Address Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04800h-04803h

DWord	Bit	Description
0	31:12	**Address** Format: GraphicsAddress[31:12] Page virtual address.
	11:0	**Reserved** Project: CHV, BSW Format: MBZ

NOP Identification Register

NOPID - NOP Identification Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Size (in bits):	32
Trusted Type:	1
Address:	02094h
Address:	12094h-12097h
Name:	NOP Identification Register
ShortName:	NOPID_VCSUNIT0
Address:	1A094h-1A097h
Name:	NOP Identification Register
ShortName:	NOPID_VECSUNIT
Address:	1C094h-1C097h
Name:	NOP Identification Register
ShortName:	NOPID_VCSUNIT1
Address:	22094h-22097h
Name:	NOP Identification Register
ShortName:	NOPID_BCSUNIT

Description	Project
Access: RW	CHV, BSW
The NOPID register contains the Noop Identification value specified by the last MI_NOOP instruction that enabled this register to be updated.	

DWord	Bit	Description
0	31:22	**Reserved** Format: MBZ
	21:0	**Reserved**

P2GCONTROL

P2GCONTROL - P2GCONTROL		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	1300F0h	
Punit to Gunit Control. Punit will context save/restore this register.		
DWord	**Bit**	**Description**
0	31:1	**Reserved** Access: RO
	0	**P2GCONTROL_POLICY** Access: R/W 0: Let Gunit do the context save. (Power-On default) 1: Context save will be handled by Punit and driver.

PAGE_FAULT_MODE

PAGE_FAULT_MODE - PAGE_FAULT_MODE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0E454h
Name:	PAGE_FAULT_MODE
ShortName:	PAGE_FAULT_MODE
Valid Projects:	CHV, BSW

This is a basic register template

DWord	Bit	Description
0	31:8	**Reserved** Default Value: 0000000000000000b Access: RO
	7:6	**FAULT_MODE** Project: CHV, BSW Access: R/W **Fault Model:** Applicable only in advanced context "00": Fault & Hang. Same mode as gen7.5 "01": Fault & Halt "10": Fault & Continue & Switch "11": Reserved

Value	Name
00b	**[Default]**
01b	

	5:0	**Reserved** Project: CHV, BSW

Page Directory Pointer Descriptor - PDP0/PML4/PASID

PDP0 - Page Directory Pointer Descriptor - PDP0/PML4/PASID

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64

Address:	02270h-02277h
Name:	Page Directory Pointer Descriptor - PDP0/PML4/PASID
ShortName:	PDP0_RCSUNIT

Address:	12270h-12277h
Name:	Page Directory Pointer Descriptor - PDP0/PML4/PASID
ShortName:	PDP0_VCSUNIT0

Address:	1A270h-1A277h
Name:	Page Directory Pointer Descriptor - PDP0/PML4/PASID
ShortName:	PDP0_VECSUNIT

Address:	1C270h-1C277h
Name:	Page Directory Pointer Descriptor - PDP0/PML4/PASID
ShortName:	PDP0_VCSUNIT1

Address:	22270h-22277h
Name:	Page Directory Pointer Descriptor - PDP0/PML4/PASID
ShortName:	PDP0_BCSUNIT

PDP0/PML4/PASID: This register can contain three values which depend on the element descriptor definition. **PASID[19:0]:** Populated in the first 20bits of the register and selected when Advanced Context flag is set in the element descriptor in execlist mode of submission. This is not valid in ring buffer mode of scheduling. **PML4[38:12]:** Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected. **PDP0[38:12]:** Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping. This is valid when Legacy Context Flag is set and 64bit virtual addressing is not supported. *Note: This is a guest physical address.*

Programming Notes	Project
Execlist Based Scheduling: SW should update PDP0/12/3 registers in context image with proper values before submitting the context to HW in execlist mode of scheduling. HW restores these registers as part of context restore to set the PPGTT access accordingly. PPGTT is always enabled in advanced context mode of execlist based scheduling and can be disabled only in legacy context mode. Privilege Access Bit in Element Descriptor controls the PPGTT enabling in legacy context mode.	

PDP0 - Page Directory Pointer Descriptor - PDP0/PML4/PASID

Ring Buffer Based Scheduling: A write via MMIO to PDP0_DESCRIPROR (lower Dword) triggers the Page Directory Restore in HW when PPGTT is enabled. SW should ensure PDP1/2/3 registers are programmed appropriately prior to programming PDP0. PDP0_DESCRIPTOR lower dword should be programmed at the end. Per-Process GTT Enable Bit in GFX_MODE register controls the PPGTT enabling and disabling. Programming Per-Process GTT Enable Bit in GFX_MODE register doesn't enable/disable the PPGTT translation of memory access immediately; the change comes in to affect only when the Page Directory registers are programmed. Programming Per-Process GTT Enable Bit in GFX_MODE register bit must be followed by programming Page Directory Registers in order to enable or disable the PPGTT translation of memory access. PDP* DESCRIPTOR registers must always be programmed through MI_LOAD_REGISTER_IMMEDIATE command in ring buffer with PDP0_DESCRIPRTOR lower dword written at the end. PDP0/12/3 registers are context save restored. PDP descriptors are power context save restored in VCS, BCS and VECS engines. PDP descriptors are context save restored per render context in RCS and must be programmed following MI_SET_CONTEXT command, in case of PDP descriptors programmed without context set (MI_SET_CONTEXT) will get lost on C6 entry/exit. PDP descriptor registers should be programmed after ensuring the pipe is completely flushed and TLB's invalidated.	CHV, BSW

DWord	Bit	Description
0	63	**PD Load Busy** Project: CHV, BSW Access: RO Format: Valid This read-only field gets set when PDP0 is written to indicating Page Directory Restore activity is in progress and will get reset once the activity is completed.
	62:0	**PDP0 Descriptor** Project: CHV, BSW

Page Directory Pointer Descriptor - PDP1

PDP1 - Page Directory Pointer Descriptor - PDP1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02278h-0227Fh
Name:	Page Directory Pointer Descriptor - PDP1
ShortName:	PDP1_RCSUNIT
Address:	12278h-1227Fh
Name:	Page Directory Pointer Descriptor - PDP1
ShortName:	PDP1_VCSUNIT0
Address:	1A278h-1A27Fh
Name:	Page Directory Pointer Descriptor - PDP1
ShortName:	PDP1_VECSUNIT
Address:	1C278h-1C27Fh
Name:	Page Directory Pointer Descriptor - PDP1
ShortName:	PDP1_VCSUNIT1
Address:	22278h-2227Fh
Name:	Page Directory Pointer Descriptor - PDP1
ShortName:	PDP1_BCSUNIT

PDP1[38:12]: Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping. This is valid when Legacy Context Flag is set and 64bit virtual addressing is not supported. *Note: This is a guest physical address.*

DWord	Bit	Description
0	63:0	**PDP1 Descriptor**

Page Directory Pointer Descriptor - PDP2

PDP2 - Page Directory Pointer Descriptor - PDP2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02280h-02287h
Name:	Page Directory Pointer Descriptor - PDP2
ShortName:	PDP2_RCSUNIT
Address:	12280h-12287h
Name:	Page Directory Pointer Descriptor - PDP2
ShortName:	PDP2_VCSUNIT0
Address:	1A280h-1A287h
Name:	Page Directory Pointer Descriptor - PDP2
ShortName:	PDP2_VECSUNIT
Address:	1C280h-1C287h
Name:	Page Directory Pointer Descriptor - PDP2
ShortName:	PDP2_VCSUNIT1
Address:	22280h-22287h
Name:	Page Directory Pointer Descriptor - PDP2
ShortName:	PDP2_BCSUNIT

PDP2[38:12]: Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping. This is valid when Legacy Context Flag is set and 64bit virtual addressing is not supported.
Note: This is a guest physical address.

DWord	Bit	Description
0	63:0	**PDP2 Descriptor**

Page Directory Pointer Descriptor - PDP3

PDP3 - Page Directory Pointer Descriptor - PDP3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02288h-0228Fh
Name:	Page Directory Pointer Descriptor - PDP3
ShortName:	PDP3_RCSUNIT
Address:	12288h-1228Fh
Name:	Page Directory Pointer Descriptor - PDP3
ShortName:	PDP3_VCSUNIT0
Address:	1A288h-1A28Fh
Name:	Page Directory Pointer Descriptor - PDP3
ShortName:	PDP3_VECSUNIT
Address:	1C288h-1C28Fh
Name:	Page Directory Pointer Descriptor - PDP3
ShortName:	PDP3_VCSUNIT1
Address:	22288h-2228Fh
Name:	Page Directory Pointer Descriptor - PDP3
ShortName:	PDP3_BCSUNIT

PDP3[38:12]: Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping. This is valid when Legacy Context Flag is set and 64bit virtual addressing is not supported.
Note: This is a guest physical address.

DWord	Bit	Description
0	63:0	**PDP3 Descriptor**

Page Request Queue Address Register 0

PAGEREQ_QADDR_0 - Page Request Queue Address Register 0

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0D0h

Register to configure the base address and size of the page request queue.

DWord	Bit	Description
0	31:12	**Page Request Queue Base Register** Default Value: 00000h Access: R/W This field points to the base of 4KB aligned invalidation request queue. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Reads of this field return the value that was last programmed to it.
	11:3	**Reserved** Default Value: 000000000b Access: RO
	2:0	**Queue Size** Default Value: 000b Access: R/W This field specifies the size of the page request queue. A value of X in this field indicates a page request queue of (2X) 4KB pages. The number of entries in the invalidation queue is 2(X+8).

Page Request Queue Address Register 1

PAGEREQ_QADDR_1 - Page Request Queue Address Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0D4h

Register to configure the base address and size of the page request queue.

DWord	Bit	Description
0	31:0	**Page Request Queue Base Register** Default Value: 00000000h Access: R/W This field points to the base of 4KB aligned invalidation request queue. Hardware may ignore and not implement bits 63:HAW, where HAW is the host address width. Reads of this field return the value that was last programmed to it.

Page Request Queue Head Register 0

PAGEREQ_QHEAD_0 - Page Request Queue Head Register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0C0h

Register indicating the page request queue head.

DWord	Bit	Description
0	31:19	**Reserved** Default Value: 0000000000000b Access: RO
	18:4	**Queue Head** Default Value: 000000000000000b Access: R/W Specifies the offset (128-bit aligned) to the page request queue for the command that is processed next by software. GFX implementation: GFX has to read the content of the Head pointer as tail pointer gets close to it to prevent overflows in page request queue.
	3:0	**Reserved** Default Value: 0h Access: RO

Page Request Queue Head Register 1

PAGEREQ_QHEAD_1 - Page Request Queue Head Register 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0C4h

Register indicating the page request queue head.

DWord	Bit	Description
0	31:0	**Page Request Queue Head Register 1 Reserved** Default Value: 00000000h Access: RO Bit[63:32]: Reserved.

Page Request Queue Tail Register 0

PAGEREQ_QTAIL_0 - Page Request Queue Tail Register 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0C8h

Register indicating the page request queue tail.

DWord	Bit	Description
0	31:1	**Queue Tail** Default Value: 00000000000000000000000000000000b Access: R/W Bit[31:19]: Reserved. Bit[18:4]: Specifies the offset (128-bit aligned) to the page request queue for the request that is written next by hardware. GFX Implementation: GT manages the tail pointer value as part of page requests. The value can be acquired as part of the RC6 exit. Bit[3:1]: Reserved.
	0	**Valid Bit** Default Value: 0b Access: R/W This bit can only be cleared by SW, which also clears the other fields.

Page Request Queue Tail Register 1

PAGEREQ_QTAIL_1 - Page Request Queue Tail Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F0CCh

Register indicating the page request queue tail.

DWord	Bit	Description
0	31:0	**Page Request Queue Tail Register 1 Reserved** Default Value: 00000000h Access: RO Bit[63:32]: Reserved.

PAK_Stream-Out Report (Errors)

PAK_ERR - PAK_Stream-Out Report (Errors)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128E8h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:22	**Reserved** Format: MBZ
	21	**Incorrect IntraMBFlag in I-slice(AVCf)**
	20	**Out of Range Symbol Code(AVC/mpeg2)**
	19	**Incorrect MBType(AVC/mpeg2**
	18	**Motion Vectors are not inside the frame boundary(mpeg2)**
	17	**Scale code is zero(mpeg2)**
	16	**Incorrect DCTtype for given motionType(mpeg2)**
	15:8	**MB Y-position** This field indicates Macro Block(MB) Y- position where an error occured while encoding.
	7:0	**MB X-position** This field indicates Macro Block(MB) X- position where an error occured while encoding.

PAK_Stream-Out Report (Warnings)

PAK_WARN - PAK_Stream-Out Report (Warnings)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128E4h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:22	**Reserved** Project: All Format: MBZ
	21	**Skip Run > 8192 (AVC)**
	20	**Incorrect SkipMB (AVC and mpeg2)**
	19	**Incorrect MV difference for dual-prime MB (mpeg2)**
	18	**End of Slice signal missing on last MB of a Row(mpeg2)**
	17	**Incorrect DCT type for field picture**
	16	**MVs are not within defined range by fcode**
	15:8	**MB Y-position**
	7:0	**MB X-position**

PAK Report Running Status

PAK_REPORT_STAT - PAK Report Running Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128ECh
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:1	**Reserved**
	0	**PAK Status** Value / Name / Description 0 / / PAK engine is IDLE 1 / / PAK engine is currently generating bit stream.

PAT Index

PAT_INDEX - PAT Index		
Register Space:	MMIO: 0/2/0	
Source:	PRM	
Default Value:	0x00000003	
Access:	R/W	
Size (in bits):	32	

DWord	Bit	Description
0	31:10	**Reserved** Default Value: 00000000000000000000b
	9:8	**Class of Service** Default Value: 00b This field controls the Class of Service sent to the LLC to determine which sub-set of Ways the surface will be stored in. The allocation of certain LLC ways to different class of service settings is a project dependent decision and listed in the PRM 00: Class0 01: Class1 10: Class2 11: Class3
	7:6	**Reserved** Default Value: 00b
	5:4	**LRU AGE** Default Value: 00b 00: Take the age value from Uncore CRs 01: Assign the age of "0" 10: Do not change the age on a hit 11: Assign the age of "3"
	3:2	**Target Cache** Default Value: 00b 00: eLLC only 01: LLC only 10: LLC/eLLC allowed 11: LLC/eLLC allowed
	1:0	**Mem Type** Default Value: 11b 00: Uncacheable(UC) 01: Write Combining(WC) 10: Write through(WT) 11: Write back(WB)

PAT Index High

PAT_INDEX_H - PAT Index High

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x03030303
Size (in bits):	32
Address:	040E4h

DWord	Bit	Description
0	31:0	**PAT Index High** Default Value: 03030303h Access: R/W **Description** Bit[31:24]: PAT Index#7: Index#7 definition for page tables. (See bit[7:0] for definition.) Bit[23:16]: PAT Index#6: Index#6 definition for page tables. (See bit[7:0] for definition.) Bit[15:8]: PAT Index#5: Index#5 definition for page tables. (See bit[7:0] for definition.) Bit[7:0]: PAT Index#4: Index#4 definition for page tables. Bit[7]: Reserved. Bit[6]: Snoop Required [CHV, BSW Only] 1: System agent will snoop the IA cores 0: System agent will not snoop the IA cores Bit[5:4]: (See below.) 00b: Age is 0. 01b: Age is 1. 10b: Age is 2. 11b: Age is 3. Bit[3:2]: (See below.) 00b: eLLC only. 01b: LLC only. 10b: LLC and eLLC allowed. 11b: L3, LLC, and eLLC are allowed. Bit[1:0]: (see below): 00b: Uncacheable (UC). 01b: Write Combining (WC). 10b: Write Through (WT). 11b: Write Back (WB).

PAT Index Low

PAT_INDEX_L - PAT Index Low

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x03030303
Size (in bits):	32
Address:	040E0h

DWord	Bit	Description
0	31:0	**PAT Index Low** Default Value: 03030303h Access: R/W **Description** Bit[31:24]: PAT Index#3: Index#3 definition for page tables. (See bit[7:0] for definition.) Bit[23:16]: PAT Index#2: Index#2 definition for page tables. (See bit[7:0] for definition.) Bit[15:8]: PAT Index#1: Index#1 definition for page tables. (See bit[7:0] for definition.) Bit[7:0]: PAT Index#0: Index#0 definition for page tables. Bit[7]: Reserved. Bit[6]: Snoop Required [CHV, BSW Only] 1: System agent will snoop the IA cores 0: System agent will not snoop the IA cores Bit[5:4]: (See below.) 00b: Age is 0. 01b: Age is 1. 10b: Age is 2. 11b: Age is 3. Bit[3:2]: (See below.) 00b: eLLC only. 01b: LLC only. 10b: LLC and eLLC allowed. 11b: L3, LLC, and eLLC are allowed. Bit[1:0]: (see below): 00b: Uncacheable (UC). 01b: Write Combining (WC). 10b: Write Through (WT). 11b: Write Back (WB).

PCBR

PCBR - PCBR

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182120h

This provides a base address for context save/restore of GT and Media power context to DRAM. Gunit adds the GAM provided offsets to this base register for power context reads and writes.
GTLC stores Power Context in DRAM.
The BIOS is expected to program this register and ensure proper allocation within Gfx stolen memory.

Removing bits 63:32 since this will always be below 4GB.

DWord	Bit	Description
0	31:12	**Power_Context_Address** Default Value: 0000000h Access: R/W Lock 4KB aligned address. Locked with bit 0.
	11:1	**Reserved** Default Value: 000h Access: RO Reserved
	0	**Power_Context_Register_Lock** Default Value: 0b Access: R/W Lock Writing a '1' to this register locks this bit - preventing further updates. The Power Context Address bits are also locked.

PCICMD_STS

PCICMD_STS - PCICMD_STS

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00100000
Size (in bits):	32
Address:	00004h

04h: PCI Command Register. 06h: PCI Status Register

DWord	Bit	Description
0	31:21	**RESERVED** Default Value: 000h Access: RO
	20	**CAPABILITY_LIST** Default Value: 1b Access: RO CAP: Indicates that the CAPPOINT register at 34h provides an offset into PCI Configuration Space containing a pointer to the location of the first item in the list
	19	**INTERRUPT_STATUS** Default Value: 0b Access: RO
	18:16	**RESERVED** Default Value: 000b Access: RO
	15:11	**RESERVED** Default Value: 00h
	10	**INTERRUPT_DISABLE** Default Value: 0b Access: R/W ID: 0 : Legacy interrupt message is enabled. ID: 1 : Disables legacy interrupt message generation on IOSF Sideband. Note : The interrupt status is not blocked from being reflected in PCISTS.IS. Note: MSI interrupt generation : (PCISTS.IS and PCICMD.BME and MSI_CAPID.MSIE) changes from 0 to 1. Note: Message bus interrupt assert is sent : (PCISTS.IS and ~PCICMD.ID and ~MSI_CAPID.MSIE) changes from 0 to 1. Note: Message bus interrupt de-assert is sent : (PCISTS.IS and ~PCICMD.ID and ~MSI_CAPID.MSIE) changes from 1 to 0.

PCICMD_STS - PCICMD_STS

	9:3	**RESERVED** Default Value: 00h Access: RO Reserved
	2	**BUS_MASTER_ENABLE** Default Value: 0b Access: R/W BME: (BME and MAE are observed. But context save/restore can occur.) 0 : Blocks the sending of MSI interrupts. 1 : Permits the sending of MSI interrupts
	1	**Reserved**
	0	**IO_SPACE_ENABLE** Default Value: 0b Access: R/W IOSE: 0 : I/O space is disabled. IORD and IOWr cycles will not be claimed. 1 : I/O space is enabled. VGA_IO and Gfx_IOBAR are checked. If an IORD/IOWR matches (VGA IO address range or GFX_IOBAR), the cycle will be claimed. Care should be taken in setting up GFX_IOBAR that more than 1 match is not made as this will result in unpredictable behavior. VGA_IO : Address[15:0] is checked to determine if it falls in the VGA IO range. (The VGA IO range is 03B0h - 03BBh and 03C0h - 03DFh.) Gfx_IOBAR : Address[15:3] is compared to GFX_IOBAR[15:3].

PCU INTERRUPT ENABLE REGISTER

PCU_INTERRUPT_IER - PCU INTERRUPT ENABLE REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	444ECh-444EFh

This table indicates which events are mapped to each bit of the GT interrupt 1 register.
The IER enabled PCU interrupt IIR sticky bits are ORed together
to generate PCU interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:26	**UNUSED0** Access: R/W
	25	**PCU_MAILBOX_EVT** Access: R/W PCU Pcode 2 driver mailbox event
	24	**PCU_THERMAL_EVT** Access: R/W PCU thermal event
	23:0	**UNUSED1** Access: R/W

PCU INTERRUPT IDENTITY REGISTER

PCU_INTERRUPT_IIR - PCU INTERRUPT IDENTITY REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	444E8h-444EBh

This table indicates which events are mapped to each bit of the GT interrupt 1 register.
The IER enabled PCU interrupt IIR sticky bits are ORed together
to generate PCU interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:26	**UNUSED0** Access: R/W One Clear
	25	**PCU_MAILBOX_EVT** Access: R/W One Clear PCU Pcode 2 driver mailbox event
	24	**PCU_THERMAL_EVT** Access: R/W One Clear PCU thermal event
	23:0	**UNUSED1** Access: R/W One Clear

PCU INTERRUPT MASK REGISTER

PCU_INTERRUPT_IMR - PCU INTERRUPT MASK REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x03000000
Size (in bits):	32
Address:	444E4h-444E7h

This table indicates which events are mapped to each bit of the GT interrupt 1 register.
The IER enabled PCU interrupt IIR sticky bits are ORed together
to generate PCU interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:26	**UNUSED0** Access: R/W
	25	**PCU_MAILBOX_EVT** Default Value: 1b Access: R/W PCU Pcode 2 driver mailbox event
	24	**PCU_THERMAL_EVT** Default Value: 1b Access: R/W PCU thermal event
	23:0	**UNUSED1** Access: R/W

PCU INTERRUPT STATUS REGISTER

PCU_INTERRUPT_ISR - PCU INTERRUPT STATUS REGISTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	444E0h-444E3h

This table indicates which events are mapped to each bit of the GT interrupt 1 register.
The IER enabled PCU interrupt IIR sticky bits are ORed together
to generate PCU interrupt pending bit in the master interrupt control register.

DWord	Bit	Description
0	31:26	**UNUSED0** Access: RO
	25	**PCU_MAILBOX_EVT** Access: RO PCU Pcode 2 driver mailbox event
	24	**PCU_THERMAL_EVT** Access: RO PCU thermal event
	23:0	**UNUSED1** Access: RO

Pending Head Pointer Register

UHPTR - Pending Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02134h-02137h
Name:	Pending Head Pointer Register
ShortName:	UHPTR_RCSUNIT
Address:	12134h-12137h
Name:	Pending Head Pointer Register
ShortName:	UHPTR_VCSUNIT0
Address:	1A134h-1A137h
Name:	Pending Head Pointer Register
ShortName:	UHPTR_VECSUNIT
Address:	1C134h-1C137h
Name:	Pending Head Pointer Register
ShortName:	UHPTR_VCSUNIT1
Address:	22134h-22137h
Name:	Pending Head Pointer Register
ShortName:	UHPTR_BCSUNIT

Programming Notes	Source
Once SW uses UHPTR to preempt the existing workload, should explicitly program MI_SET_CONTEXT to save the preempted context status before submitting the new workload. In case SW doesn't want to save the state of the preempted context, it should at the minimum program RS_PREEMPT_STATUS to 0x0 so that the register status doesn't interfere with the new workloads.	RenderCS

DWord	Bit	Description
0	31:3	**Head Pointer Address** Format: GraphicsAddress[31:3] **Description** This register represents the GFX address offset where execution should continue in the ring buffer following execution of a Preemptable Command. Refer to the Preemption section for the list of preemptable commands supported in ring buffer mode of scheduling.

UHPTR - Pending Head Pointer Register

	2:1	**Reserved** Format: MBZ
	0	**Head Pointer Valid** **Description** This bit is set by the software to request a pre-emption. It is reset by hardware when a Preemptable command is parsed by the command streamer. The hardware uses the head pointer programmed in this register at the time the reset is generated. Refer to the Preemption section for the list of preemptable commands supported in ring buffer mode of scheduling. This bit is treated as set by command streamer only when arbitration is not disabled using MI_ARB_ON_OFF command. Preemption will not occur on MI_ARB_CHEK command when UHPTR is valid if the arbitration is disabled using MI_ARB_ON_OFF command.

Value	Name	Description
0	InValid	No valid updated head pointer register, resume execution at the current location in the ring buffer
1	Valid	Indicates that there is an updated head pointer programmed in this register

Plink G2H Spare

SPAREG2H - Plink G2H Spare		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A194h-0A197h	
DWord	**Bit**	**Description**
0	31:9	**Reserved** Access: RO
	8:0	**Plink_G2H_Spare** Access: R/W These are spares for ECO use.

PM_PWR_CLK_STATE

PM_PWR_CLK_STATE - PM_PWR_CLK_STATE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00008288
Size (in bits):	32
Address:	0A200h

PM Power Clock State Request

DWord	Bit	Description
0	31	**Enable** Access: R/W Power Clock State Enable: 0 : No specific power state set, no message/wait with PMunit 1 : CSunit sends the contents of this register to PMunit each time it is written, Send contents of this register to PMunit, wait for Ack. When CS writes to A200, requesting new set of resources: Actual EUs used = Async_EU if EUmin < Async_EU < EUmax Actual EUs used = EUmax if Async_EU > equal to EUmax Actual EUs used = EUmin if EUmin > equal to Async_EU Actual SSs used = Async_SS if SScountEn=0 Actual SSs used = SScount if SScountEn=1 After new resources are set by GPMunit, GPM replies to CS by writing 0001_0001 to 0x00_300c
	30:19	**RSVD** Access: RO Reserved (CSunit implements full 32b storage)
	18	**SCountEn** Access: R/W Programming Notes: Not supported in CHV, BSW. Must be zero (MBZ). — Project: CHV, BSW
	17:15	**SliceCount** Default Value: 001b Access: R/W Programming Notes: Not supported in CHV, BSW. Must be zero (MBZ). — Project: CHV, BSW

PM_PWR_CLK_STATE - PM_PWR_CLK_STATE

	14:13	**RSVD** Access: RO Reserved (CSunit implements full 32b storage)
	12	**Spare** Access: R/W Spare bit for CHV, BSW Programming Notes: Must be zero (MBZ). Project: CHV, BSW
	11	**SSCountEn** Access: R/W Enable Subslice Count Request 0 = Use async PMunit subslice count request 1 = Use SliceCount from this register
	10:8	**SScount** Default Value: 010b Access: R/W Number of subslices to power: 001 : 1 subslice 010 : 2 subslices (GT1-based CHV, BSW only)
	7:4	**EUmax** Default Value: 1000b Access: R/W Maximum number of EUs to power (per subslice if multiple subslices enabled). To specify an exact number of subslices, set EUmax equal to EUmin MinEU and MaxEU need to be even and that odd numbers are illegal
	3:0	**EUmin** Default Value: 1000b Access: R/W Minimum number of EUs to power (per subslice if multiple subslices enabled): 0010 : 2 EUs 0100 : 4 EUs 0110 : 6 EUs 1000 :8 EUs MinEU and MaxEU need to be even and that odd numbers are illegal When both subslices are enabled (ie 0x18_2168[11:10] = 2'b00)

PM_PWR_CLK_STATE - PM_PWR_CLK_STATE

Config	Disabled Column	FUSE_GT_EU_DISABLE - 0x18_2168[31:16]	Desired Config	Actual/Driver Config
2x8	none	0x0000	2x8	2x8
2x8	none	0x0000	2x6	2x6
2x8	none	0x0000	2x4	2x4
2x8	none	0x0000	2x2	2x2
2x6	1	0x1111	2x8	NA
2x6	1	0x1111	2x6	2x8
2x6	1	0x1111	2x4	2x6
2x6	1	0x1111	2x2	2x4
2x6	2	0x2222	2x8	NA
2x6	2	0x2222	2x6	2x8
2x6	2	0x2222	2x4	2x6
2x6	2	0x2222	2x2	2x2
2x6	3	0x4444	2x8	NA
2x6	3	0x4444	2x6	2x8
2x6	3	0x4444	2x4	2x4
2x6	3	0x4444	2x2	2x2
2x6	4	0x8888	2x8	NA
2x6	4	0x8888	2x6	2x6
2x6	4	0x8888	2x4	2x4
2x6	4	0x8888	2x2	2x2
2x5	1+EU3	0x1919	2x8	NA
2x5	1+EU3	0x1919	2x5	2x8
2x5	1+EU3	0x1919	2x4	2x6
2x5	1+EU3	0x1919	2x2	2x4
2x5	2+EU3	0x2A2A	2x8	NA
2x5	2+EU3	0x2A2A	2x5	2x8
2x5	2+EU3	0x2A2A	2x4	2x6
2x5	2+EU3	0x2A2A	2x2	2x2

PM_PWR_CLK_STATE - PM_PWR_CLK_STATE

		2x5	3+EU3	0x4C4C	2x8	NA
		2x5	3+EU3	0x4C4C	2x5	2x8
		2x5	3+EU3	0x4C4C	2x4	2x4
		2x5	3+EU3	0x4C4C	2x2	2x2
		2x5	4+EU2	0x8C8C	2x8	NA
		2x5	4+EU2	0x8C8C	2x5	2x6
		2x5	4+EU2	0x8C8C	2x4	2x4
		2x5	4+EU2	0x8C8C	2x2	2x2

PMCAPID

PMCAPID - PMCAPID

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00229001
Size (in bits):	32
Address:	000D0h

PCI Address: D0-D1h Description : Power Management Capabilities ID
PCI Address: D2-D3h Description : Power Management Capabilities

DWord	Bit	Description
0	31:27	**PME_SUPPORT** Default Value: 00h Access: RO PMES The graphics controller does not generate PME.
	26	**D2_SUPPORT** Default Value: 0b Access: RO D2S: D2 power management state is not supported.
	25	**D1_SUPPORT** Default Value: 0b Access: RO D1S: D1 power management state is not supported.
	24:22	**RESERVED** Default Value: 000b Access: RO Reserved
	21	**DEVICE_SPECIFIC_INITIALIZATION** Default Value: 1b Access: RO Hardwired to 1 to indicate that special initialization of the graphics controller is required before generic class device driver is to use it.
	20:19	**RESERVED** Default Value: 00b

PMCAPID - PMCAPID

	Access:	RO
	Reserved	
18:16	**VERSION**	
	Default Value:	010b
	Access:	RO
	Verion compliance with revision 1.1 of PCI Power management spec.	
15:8	**NEXT_POINTER**	
	Default Value:	90h
	Access:	R/W Once
	Indicates the next item in the capabilities list(90=MSI) This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset. Write once allowing changing of the capabilities list.	
7:0	**CAPABILITIES_ID**	
	Default Value:	01h
	Access:	RO
	CAPID: SIG defines this ID is 01h for power management.	

PMCS

PMCS - PMCS

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	000D4h

Power Management Control/Status. Driver does not use this register. SBIOS does not use this register

DWord	Bit	Description
0	31:2	**RESERVED** Default Value: 00000000h Access: RO Reserved
	1:0	**POWER_STATE_PS** Default Value: 00b Access: R/W This field indicates the current power state of the IGD and can be used to set the IGD into a new power state. If software attempts to write an unsupported state to this field, write operation must complete normally on the bus, but the data is discarded and no state change occurs. On a transition from D3 to D0 the graphics controller is optionally reset to initial values. Behavior of the graphics controller in supported states is detailed in the power management section of the PRM. Bits[1:0] Power state 00: D0 Default 01: D1 Not Supported 10: D2 Not Supported 11: D3

POWER_WELL_SS0_SIG1

POWER_WELL_SS0_SIG1 - POWER_WELL_SS0_SIG1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x02020242
Size (in bits):	32
Address:	0A720h

Subwell signals to be driven out to subslice0 wells: Subwells 1-4.
This register is modified by HW based on sub-power domain configuration and SW should not be writing to this register.

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b Access: RO Reserved
	29	**ss0_eu210_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss0_eu210'
	28	**ss0_eu210_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss0_eu210'
	27	**ss0_eu210_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss0_eu210'
	26	**ss0_eu210_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss0_eu210'
	25	**ss0_eu210_pgenb**

POWER_WELL_SS0_SIG1 - POWER_WELL_SS0_SIG1

		Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss0_eu210' 0 = sub-well is powered on. 1 = sub-well is powered off.
	24	**ss0_eu210_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss0_eu210'
	23:22	**Reserved** Default Value: 00b Access: RO Reserved
	21	**ss0_eu19_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss0_eu19'
	20	**ss0_eu19_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss0_eu19'
	19	**ss0_eu19_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss0_eu19'
	18	**ss0_eu19_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss0_eu19'
	17	**ss0_eu19_pgenb** Default Value: 1b Access: R/W

POWER_WELL_SS0_SIG1 - POWER_WELL_SS0_SIG1

		Subwell control signal 'pgenb' for subwell 'ss0_eu19' 0 = sub-well is powered on. 1 = sub-well is powered off.
	16	**ss0_eu19_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss0_eu19'
	15:14	**Reserved** Default Value: 00b Access: RO Reserved
	13	**ss0_eu08_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss0_eu08'
	12	**ss0_eu08_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss0_eu08'
	11	**ss0_eu08_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss0_eu08'
	10	**ss0_eu08_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss0_eu08'
	9	**ss0_eu08_pgenb** Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss0_eu08' 0 = sub-well is powered on. 1 = sub-well is powered off.

POWER_WELL_SS0_SIG1 - POWER_WELL_SS0_SIG1

8	**ss0_eu08_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss0_eu08'
7	**Reserved** Default Value: 0b Access: RO Reserved
6	**ss0_pgenb_sig1** Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss0_ss0'
5	**ss0_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss0_ss0'
4	**ss0_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss0_ss0'
3	**ss0_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss0_ss0'
2	**ss0_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss0_ss0'
1	**ss0_pgenb_sig0** Default Value: 1b Access: R/W

POWER_WELL_SS0_SIG1 - POWER_WELL_SS0_SIG1		
		Subwell control signal 'pgenb' for subwell 'ss0_ss0' 0 = sub-well is powered on. 1 = sub-well is powered off.
	0	**ss0_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss0_ss0'

POWER_WELL_SS0_SIG2

POWER_WELL_SS0_SIG2 - POWER_WELL_SS0_SIG2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000002
Size (in bits):	32
Address:	0A724h

Subwell signals to be driven out to subslice0 wells: subwell 5.
This register is modified by HW based on sub-power domain configuration and SW should not be writing to this register.

DWord	Bit	Description
0	31:6	**Reserved** Default Value: 0000000h Access: RO Reserved
	5	**ss0_eu311_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss0_eu311'
	4	**ss0_eu311_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss0_eu311'
	3	**ss0_eu311_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss0_eu311'
	2	**ss0_eu311_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss0_eu311'
	1	**ss0_eu311_pgenb**

POWER_WELL_SS0_SIG2 - POWER_WELL_SS0_SIG2

		Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss0_eu311' 0 = sub-well is powered on. 1 = sub-well is powered off.
	0	**ss0_eu311_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss0_eu311'

POWER_WELL_SS1_SIG1

POWER_WELL_SS1_SIG1 - POWER_WELL_SS1_SIG1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x02020242
Size (in bits):	32
Address:	0A728h

Subwell signals to be driven out to subslice1 wells: Subwells 1-4.
This register is modified by HW based on sub-power domain configuration and SW should not be writing to this register.

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b Access: RO Reserved
	29	**ss1_eu210_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss1_eu210'
	28	**ss1_eu210_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss1_eu210'
	27	**ss1_eu210_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss1_eu210'
	26	**ss1_eu210_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss1_eu210'
	25	**ss1_eu210_pgenb**

POWER_WELL_SS1_SIG1 - POWER_WELL_SS1_SIG1

		Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss1_eu210' 0 = sub-well is powered on. 1 = sub-well is powered off.
	24	**ss1_eu210_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss1_eu210'
	23:22	**Reserved** Default Value: 00b Access: RO Reserved
	21	**ss1_eu19_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b ' for subwell 'ss1_eu19'
	20	**ss1_eu19_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss1_eu19'
	19	**ss1_eu19_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss1_eu19'
	18	**ss1_eu19_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss1_eu19'
	17	**ss1_eu19_pgenb** Default Value: 1b Access: R/W

POWER_WELL_SS1_SIG1 - POWER_WELL_SS1_SIG1

		Subwell control signal 'pgenb' for subwell 'ss1_eu19' 0 = sub-well is powered on. 1 = sub-well is powered off.
	16	**ss1_eu19_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss1_eu19'
	15:14	**Reserved** Default Value: 00b Access: RO Reserved
	13	**ss1_eu08_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b ' for subwell 'ss1_eu08'
	12	**ss1_eu08_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss1_eu08'
	11	**ss1_eu08_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss1_eu08'
	10	**ss1_eu08_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss1_eu08'
	9	**ss1_eu08_pgenb** Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss1_eu08' 0 = sub-well is powered on. 1 = sub-well is powered off.

POWER_WELL_SS1_SIG1 - POWER_WELL_SS1_SIG1

8	**ss1_eu08_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss1_eu08'
7	**Reserved** Default Value: 0b Access: RO Reserved
6	**ss1_pgenb_sig1** Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss1_ss1'
5	**ss1_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b ' for subwell 'ss1_ss1'
4	**ss1_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss1_ss1'
3	**ss1_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss1_ss1'
2	**ss1_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss1_ss1'
1	**ss1_pgenb_sig0** Default Value: 1b Access: R/W

POWER_WELL_SS1_SIG1 - POWER_WELL_SS1_SIG1

		Subwell control signal 'pgenb' for subwell 'ss1_ss1' 0 = sub-well is powered on. 1 = sub-well is powered off.
	0	**ss1_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss1_ss1'

POWER_WELL_SS1_SIG2

POWER_WELL_SS1_SIG2 - POWER_WELL_SS1_SIG2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000002
Size (in bits):	32
Address:	0A72Ch

Subwell signals to be driven out to subslice1 wells: subwell 5.
This register is modified by HW based on sub-power domain configuration and SW should not be writing to this register.

DWord	Bit	Description
0	31:6	**Reserved** Default Value: 0000000h Access: RO Reserved
	5	**ss1_eu311_rst_b** Default Value: 0b Access: R/W Subwell control signal 'rst_b' for subwell 'ss1_eu311'
	4	**ss1_eu311_fwenb** Default Value: 0b Access: R/W Subwell control signal 'fwenb' for subwell 'ss1_eu311'
	3	**ss1_eu311_pwrok** Default Value: 0b Access: R/W Subwell control signal 'pwrok' for subwell 'ss1_eu311'
	2	**ss1_eu311_asyncrst_b** Default Value: 0b Access: R/W Subwell control signal 'asyncrst_b' for subwell 'ss1_eu311'

POWER_WELL_SS1_SIG2 - POWER_WELL_SS1_SIG2

	1	**ss1_eu311_pgenb** Default Value: 1b Access: R/W Subwell control signal 'pgenb' for subwell 'ss1_eu311' 0 = sub-well is powered on. 1 = sub-well is powered off.
	0	**ss1_eu311_clocken** Default Value: 0b Access: R/W Subwell control signal 'clocken' for subwell 'ss1_eu311'

Power Context Save

PWRCTXSAVE - Power Context Save

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04A04h

DWord	Bit	Description
0	31:16	**Mask Bits** Default Value: 0000h Access: RO
	15	**Extra Bits15** Default Value: 0b Access: R/W Extra Bits for future use.
	14	**Extra Bits14** Default Value: 0b Access: R/W Extra Bits for future use.
	13	**Extra Bits13** Default Value: 0b Access: R/W Extra Bits for future use.
	12	**Extra Bits12** Default Value: 0b Access: R/W Extra Bits for future use.
	11	**Extra Bits11** Default Value: 0b Access: R/W Extra Bits for future use.
	10	**Extra Bits10** Default Value: 0b Access: R/W Extra Bits for future use.

PWRCTXSAVE - Power Context Save

	9	**Power Context Save Request** Default Value: 0b Access: R/W Power Context Save. Bit[9].Power Context Save Request. 1'b0: Power context save is not being requested (default). 1'b1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. This bit is self clear.
	8:0	**Power Context Save Quad Word Credits** Default Value: 000000000b Access: R/W Power Context Save. Bits[8:0].QWord Credits for Power Context Save Request. An initial length packet is required per power context save session, but that packet does not consume a credit. See protocol description for more details. Minimum Credits = 1: Unit may send 1 QWord pair. Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Only valid with PWRCTX_SAVE_REQ (Bit9).

Power context Save Register for LPFC

LPCSR - Power context Save Register for LPFC		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B408h	

DWord	Bit	Description
0	31:10	**Reserved** Access: RO Reserved.
	9:0	**Power context save register command** Access: R/W Hardware Clear Bit[9].Power Context Save Request. 1'b0: Power context save is not being requested (default). 1'b1: Power context save is being requested. Unit needs to self-clear this bit upon sampling. Bits[8:0].QWord Credits for Power Context Save Request. Minimum Credits = 1: Unit may send 1 QWord pair (enough for first LRI at least). Maximum Credits = 511: Unit may send 511 QWord pairs. A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

Power Context Save request

PCTXSAVEREQ - Power Context Save request		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	08110h	

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask bots for lower 16 bits
	15:10	**Reserved** Access: RO Reserved
	9	**Power context save req** Access: R/W Set Power Context Save Request 1'b0 : Power context save is not being requested (default) 1'b1 : Power context save is being requested CPUnit self-clears this bit upon sampling.
	8:0	**Power Context Save request credit count** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTX_SAVE_REQ (Bit9).

POWERDOWN_STATE

POWERDOWN_STATE - POWERDOWN_STATE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x66654321
Size (in bits):	32
Address:	0A710h

Subwell Power down state description.

DWord	Bit	Description
0	31:28	**pmcr_powerdown_state8** Default Value: 0110b Access: R/W The 8th state/signal that needs to be driven
	27:24	**pmcr_powerdown_state7** Default Value: 0110b Access: R/W The 7th state/signal that needs to be driven
	23:20	**pmcr_powerdown_state6** Default Value: 0110b Access: R/W The 6th state/signal that needs to be driven
	19:16	**pmcr_powerdown_state5** Default Value: 0101b Access: R/W The 5th state/signal that needs to be driven
	15:12	**pmcr_powerdown_state4** Default Value: 0100b Access: R/W The 4th state/signal that needs to be driven
	11:8	**pmcr_powerdown_state3** Default Value: 0011b Access: R/W

POWERDOWN_STATE - POWERDOWN_STATE

		The 3rd state/signal that needs to be driven
	7:4	**pmcr_powerdown_state2** Default Value: 0010b Access: R/W The 2nd state/signal that needs to be driven
	3:0	**pmcr_powerdown_state1** Default Value: 0001b Access: R/W This is the very first state and the signal that needs to be driven appropriately Encodings: 001 = Reset Sync Reset 010 = Reset Firewall 011 = Set Power EnB 100 = Reset Clock En 101 = Reset Async Reset 110 = Done 111 = Reserved. Open: Do we want to have an extra bit for future support

POWERDOWN_WAIT1

POWERDOWN_WAIT1 - POWERDOWN_WAIT1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x01010101	
Size (in bits):	32	
Address:	0A714h	
Subwell power down state wait time 1.		
DWord	**Bit**	**Description**
0	31:24	**pmcr_powerdown_wait_state4_5** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 4 is driven before signal 5 can be driven
	23:16	**pmcr_powerdown_wait_state3_4** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 3 is driven before signal 4 can be driven
	15:8	**pmcr_powerdown_wait_state2_3** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 2 is driven before signal 3 can be driven
	7:0	**pmcr_powerdown_wait_state1_2** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 1 is driven before signal 2 can be driven. Note: The granularity of these are in 30ns intervals meaning the counters will increment/decrement every 30ns pulse

POWERDOWN_WAIT2

POWERDOWN_WAIT2 - POWERDOWN_WAIT2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01010101
Size (in bits):	32
Address:	0A718h

Subwell power down state wait time 2.

DWord	Bit	Description
0	31:24	**pmcr_powerdown_interval** Default Value: 01h Access: R/W Stagger between different power gate enables for power down.
	23:16	**pmcr_powerdown_wait_state7_8** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 7 is driven before signal 8 can be driven
	15:8	**pmcr_powerdown_wait_state6_7** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 6 is driven before signal 7 can be driven
	7:0	**pmcr_powerdown_wait_state5_6** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 5 is driven before signal 6 can be driven

Power Down/Up Control

PWRDWNUPCTL - Power Down/Up Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A294h-0A297h

DWord	Bit	Description
0	31:4	**Reserved** Access: RO
	3	**Block Signaling Policy** Access: R/W BlockAll signaling policy register. Applies to signaling sent to Wake FIFO (ie. IA GT FIFO and HW FIFO). 0 (default) : BlockALL signal only asserts for CPD. 1 : BlockALL signal can assert for CPD and for a timeperiod during RC6 entry. Note : This affects MMIO requests crossing from CZ clock domain to message channel in Gfx clock domain.
	2	**Serialize Power Requests** Access: R/W Determines whether simultaneous requests to REMOVE power are presented to the Punit at the same time or whether the handshake for one must complete before the handshake for the other may start. 0=simultaneous 1=serial
	1	**Powerdown Request Order** Access: R/W Determines which well is disabled first if both wells are requesting to enter standby simultaneously, if serialized. 0=render well power is removed first 1=media well power is removed first.
	0	**Wake Media First** Access: R/W When waking both wells, it determines which well wakes first. 0=wake render first 1=wake media first. (It is NOT possible to wake both simultaneously for di/dt reasons)

Power Enable stagger control

PWRENSTAGCNTRL - Power Enable stagger control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001
Size (in bits):	32
Address:	0A70Ch

Power Enable Stagger Control

DWord	Bit	Description
0	31:8	**Reserved** Access: RO
	7:0	**pgen_phase_interval** Default Value: 01h Access: R/W It is the separation between pgen0 and pgen1 on SSx. (units = 30ns) Default value 0x01. (Real silicon settings will be determined by circuit simulation, but expect it to be no higher than 0x21. Nominally, expect it to be 0x0A)

Power Meter Weight for gti_idle_cz

PMWGICZ - Power Meter Weight for gti_idle_cz

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300A4h

This register contains the power meter weight for the idleness in CZ clock domain.

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved
	15:0	**PMWGICZ** Default Value: 0000h Access: R/W This 16-bit value is used to indicate how often to accumulate power within the CZ clock domain. When this field is set to zero, CZ clock domain will not contribute to the overall energy count. Otherwise, the value in this register will dictate how often to add CZ power contribution.

Power Meter Weight for gti_idle_gs

PWRMTR_WT_GTI - Power Meter Weight for gti_idle_gs	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AABCh

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:0	**Power Meter Weight GTI** Access: R/W Power meter weight for gti_idle_gs.

Power Meter Weight for media/render

PWRMTR_WT_MEDREN - Power Meter Weight for media/render

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AAB8h

DWord	Bit	Description
0	31:16	**Power Meter Weight Render** Access: R/W Power meter weight for render_idle.
	15:0	**Power Meter Weight Media** Access: R/W Power meter event weight media_idle.

POWERUP_STATE

POWERUP_STATE - POWERUP_STATE

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x76543210
Size (in bits):	32
Address:	0A700h

Subwell Power up state description.

DWord	Bit	Description
0	31:28	**pmcr_powerup_state8** Default Value: 0111b Access: R/W The 8th state/signal that needs to be driven
	27:24	**pmcr_powerup_state7** Default Value: 0110b Access: R/W The 7th state/signal that needs to be driven
	23:20	**pmcr_powerup_state6** Default Value: 0101b Access: R/W The 6th state/signal that needs to be driven
	19:16	**pmcr_powerup_state5** Default Value: 0100b Access: R/W The 5th state/signal that needs to be driven
	15:12	**pmcr_powerup_state4** Default Value: 0011b Access: R/W The 4th state/signal that needs to be driven
	11:8	**pmcr_powerup_state3** Default Value: 0010b Access: R/W

POWERUP_STATE - POWERUP_STATE

		The 3rd state/signal that needs to be driven
	7:4	**pmcr_powerup_state2** Default Value: 0001b Access: R/W The 2nd state/signal that needs to be driven
	3:0	**pmcr_powerup_state1** Default Value: 0000b Access: R/W This is the very first state and the signal that needs to be driven appropriately Encodings: 0000 = First Set Clock En 0001 = Reset Power EnB 0010 = Reset Clock En 0011 = Set Firewall 0100 = Set Async Reset/ Set Pwrok 0101 = Final Set clock en 0110 = Set Sync Reset. 0111 = Done 1xxx = Reserved for future

POWERUP_WAIT1

POWERUP_WAIT1 - POWERUP_WAIT1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01010101
Size (in bits):	32
Address:	0A704h

Subwell power up state wait time 1

DWord	Bit	Description
0	31:24	**pmcr_powerup_wait_state4_5** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 4 is driven before signal 5 can be driven
	23:16	**pmcr_powerup_wait_state3_4** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 3 is driven before signal 4 can be driven
	15:8	**pmcr_powerup_wait_state2_3** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 2 is driven before signal 3 can be driven
	7:0	**pmcr_powerup_wait_state1_2** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 1 is driven before signal 2 can be driven. Note: The granularity of these are in 30ns intervals meaning the counters will increment or decrement every 30ns pulse unless we are asserting clocken in which case it is on a usync boundary

POWERUP_WAIT2

POWERUP_WAIT2 - POWERUP_WAIT2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01010101
Size (in bits):	32
Address:	0A708h

Subwell power up state wait time 2.

DWord	Bit	Description
0	31:24	**pmcr_powerup_interval** Default Value: 01h Access: R/W Stagger between different power gate enables on power up
	23:16	**pmcr_powerup_wait_state7_8** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 7 is driven before signal 8 can be driven
	15:8	**pmcr_powerup_wait_state6_7** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 6 is driven before signal 7 can be driven
	7:0	**pmcr_powerup_wait_state5_6** Default Value: 01h Access: R/W This is the programmed value as to how long there is a wait from the time signal 5 is driven before signal 6 can be driven

PPGTT Page Fault Data Registers

PP_PFD[0:31] - PPGTT Page Fault Data Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	04580h

The GTT Page Fault Log entries can be read from these registers.
4580h-4583h: Fault Entry 0
...
45FCh-45FFh: Fault Entry 31

DWord	Bit	Description
0	31:12	**Fault Entry Page Address** Format: GraphicsAddress[31:12] This RO field contains the faulting page address for this Fault Log entry. This field will contain a valid fault address only if the bit in the GTT Page Fault Indication Register corresponding with the address offset of this entry is set.
	11:0	**Reserved** Format: MBZ

Predicate Rendering Data Result

MI_PREDICATE_RESULT - Predicate Rendering Data Result	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02418h
Valid Projects:	

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**MI_PREDICATE_RESULT** This bit is the result of the last MI_PREDICATE.

Predicate Rendering Data Result 1

MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0241Ch-0241Fh
Name:	Predicate Rendering Data Result 1
ShortName:	MI_PREDICATE_RESULT_1_RCSUNIT
Address:	1241Ch-1241Fh
Name:	Predicate Rendering Data Result 1
ShortName:	MI_PREDICATE_RESULT_1_VCSUNIT0
Address:	1A41Ch-1A41Fh
Name:	Predicate Rendering Data Result 1
ShortName:	MI_PREDICATE_RESULT_1_VECSUNIT
Address:	1C41Ch-1C41Fh
Name:	Predicate Rendering Data Result 1
ShortName:	MI_PREDICATE_RESULT_1_VCSUNIT1
Address:	2241Ch-2241Fh
Name:	Predicate Rendering Data Result 1
ShortName:	MI_PREDICATE_RESULT_1_BCSUNIT

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**MI_PREDICATE_RESULT_1** This bit is used to predicate MI_BATCH_BUFFER_START commands in the RCS command stream. Usage Model: MI_MATH command will be used to do some ALU operations over GPR followed by a MI_LOAD_REGISTER_REGISTER to move the result from GPR to MI_PREDICATE_RESULT_1.

Predicate Rendering Data Result 2

MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	023BCh-023BFh
Name:	Predicate Rendering Data Result 2
ShortName:	MI_PREDICATE_RESULT_2_RCSUNIT
Valid Projects:	CHV, BSW

Address:	123BCh-123BFh
Name:	Predicate Rendering Data Result 2
ShortName:	MI_PREDICATE_RESULT_2_VCSUNIT0
Valid Projects:	CHV, BSW

Address:	1A3BCh-1A3BFh
Name:	Predicate Rendering Data Result 2
ShortName:	MI_PREDICATE_RESULT_2_VECSUNIT
Valid Projects:	CHV, BSW

Address:	1C3BCh-1C3BFh
Name:	Predicate Rendering Data Result 2
ShortName:	MI_PREDICATE_RESULT_2_VCSUNIT1
Valid Projects:	CHV, BSW

Address:	223BCh-223BFh
Name:	Predicate Rendering Data Result 2
ShortName:	MI_PREDICATE_RESULT_2_BCSUNIT

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**MI_PREDICATE_RESULT_2** This bit must be loaded with by SW based on GT mode of operation. This register must be loaded appropriately before using MI_SET_PREDICATE command. Value / Name / Description 0h / **[Default]** / Indicates GT2 mode and lower slice is disabled. 1h / / Indicates GT3 mode and lower slice is enabled.

Predicate Rendering Data Storage

MI_PREDICATE_DATA - Predicate Rendering Data Storage

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02410h-02417h
Valid Projects:	

DWord	Bit	Description
0	63:32	**MI_PREDICATE_DATA_UDW** This register is used either as computed value based off the MI_PREDICATE_SRC0 and MI_PREDICATE_SRC1 or a temporary register. See Predicate Rendering section for more details.
	31:0	**MI_PREDICATE_DATA_LDW** This register is used either as computed value based off the MI_PREDICATE_SRC0 and MI_PREDICATE_SRC1 or a temporary register. See Predicate Rendering section for more details.

Predicate Rendering Temporary Register0

MI_PREDICATE_SRC0 - Predicate Rendering Temporary Register0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02400h-02407h
Valid Projects:	

DWord	Bit	Description
0	63:0	**MI_PREDICATE_SRC0** This register is a temporary register for Predicate Rendering. See Predicate Rendering section for more details.

Predicate Rendering Temporary Register1

MI_PREDICATE_SRC1 - Predicate Rendering Temporary Register1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	02408h-0240Fh
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	63:0	**MI_PREDICATE_SRC1** This register is a temporary register for Predicate Rendering. See Predicate Rendering section for more details.

Previous Idle/Busy/Avg Count for Freq Down Recommendation

RPPREVDN - Previous Idle/Busy/Avg Count for Freq Down Recommendation

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A064h-0A067h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO Reserved
	23:0	**Previous Busy in Down EI** Access: RO Previous Busy in Down EI (PRVBSYTAVG): Reports the busyness at the end of the previous down evaluation interval 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_previous_ei_down_busy[23:0]

Previous Idle/Busy/Avg Count for Freq Up Recommendation

RPPREVUP - Previous Idle/Busy/Avg Count for Freq Up Recommendation

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A058h-0A05Bh

DWord	Bit	Description
0	31:24	**Reserved** Access: RO Reserved
	23:0	**Previous Busy in UP EI** Access: RO Reports the busyness at the end of the previous Up evaluation interval 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_previous_ei_up_busy[23:0]

Primitives Generated By VF

IA_PRIMITIVES_COUNT - Primitives Generated By VF	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02318h
Valid Projects:	

This register stores the count of primitives generated by VF. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**IA Primitives Count Report UDW** Total number of primitives output by the Vertex Fetch (IA) stage. This count is updated for every primitive output by the VF stage, as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)
	31:0	**IA Primitives Count Report LDW** Total number of primitives output by the Vertex Fetch (IA) stage. This count is updated for every primitive output by the VF stage, as long as Statistics Enable is set in VF_STATE (see the Vertex Fetch Chapter in the 3D Volume.)

Private PAT

PRIV_PAT - Private PAT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000003
Size (in bits):	32
Address:	040E8h

DWord	Bit	Description
0	31:0	**Private PAT** Default Value: 00000003h Access: R/W

Description	Project
Bit[31:8]: Reserved.	
Bit[7]: Reserved. Bit[6]: Snoop Required [CHV, BSW Only] 1: System agent will snoop the IA cores 0: System agent will not snoop the IA cores Bit[5:4]: (See below.) 00b: Age is 0. 01b: Age is 1. 10b: Age is 2. 11b: Age is 3. Bit[3:2]: (See below.) 00b: eLLC only. 01b: LLC only. 10b: LLC and eLLC allowed. 11b: L3, LLC, and eLLC are allowed. Bit[1:0]: (see below): 00b: Uncacheable (UC). 01b: Write Combining (WC). 10b: Write Through (WT). 11b: Write Back (WB).	CHV, BSW

Private PAT

PRIV_PAT - Private PAT		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	040E8h	

DWord	Bit	Description
0	31:0	**Private PAT** Default Value: 00000000h Access: R/W Bit[31:16]: Reserved. Bit[15:8]: PPGTT Private PAT. (See bit[7:0] for definition.) Bit[7:6]: Reserved. Bit[5:4]: (See below.) 00b: Age is 0. 01b: Age is 1. 10b: Age is 2. 11b: Age is 3. Bit[3:2]: (See below.) 00b: Override to eLLC Only. (This setting overrides the memory_object_control_state via surface state to be eLLC target only.) 01b: eLLC only. 10b: LLC only. 11b: eLLC/LLC. Bit[1:0]: (see below): 00b: Uncached with fence. 01b: Write Combining (traditional UC). 10b: Write Through. 11b: Write Back.

PS Depth Count

PS_DEPTH_COUNT - PS Depth Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02350h

This register stores the value of the count of samples that have passed the depth test. This register is part of the context save and restore. Note that the value of this register can be obtained in a pipeline-synchronous fashion without a pipeline flush by using the 3DCONTROL command. See 3D Overview in the 3D volume.

DWord	Bit	Description
0..1	63:32	**Depth Count UDW** This register reflects the total number of samples that have passed the depth test (i.e., will be visible). All samples are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Samples that pass the depth test but fail the stencil test will not be counted.
	31:0	**Depth Count LDW** This register reflects the total number of samples that have passed the depth test (i.e., will be visible). All samples are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Samples that pass the depth test but fail the stencil test will not be counted.

PS Depth Count for Slice0

PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	022D8h

This register stores the value of the count of pixels that have passed the depth test in Slice0. This register is part of the context save and restore. This register should not be programmed by SW.Note: In GT3 mode there are two slices Slice0 (Upper Slice) and Slice1 (Lower slice).

DWord	Bit	Description
0..1	63:32	**Depth Count UDW** This register reflects the total number of pixels that have passed the depth test in Slice0(i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.
	31:0	**Depth Count LDW** This register reflects the total number of pixels that have passed the depth test in Slice0(i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.

PS Depth Count for Slice1

PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	022F8h

This register stores the value of the count of pixels that have passed the depth test in Slice1. This register is part of the context save and restore. This register should not be programmed by SW.Note: In GT3 mode there are two slices Slice0 (Upper Slice) and Slice1 (Lower slice).

DWord	Bit	Description
0..1	63:32	**Depth Count UDW** This register reflects the total number of pixels that have passed the depth test in Slice1(i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.
	31:0	**Depth Count LDW** This register reflects the total number of pixels that have passed the depth test in Slice1(i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.

PS Depth Count for Slice2

PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02450h

This register stores the value of the count of pixels that have passed the depth test in Slice2. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0..1	63:32	**Depth Count UDW** This register reflects the depth test in slice2 (i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.
	31:0	**Depth Count LDW** This register reflects the depth test in slice2 (i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.

PS Depth Count for Slice3

PS_DEPTH_COUNT_SLICE3 - PS Depth Count for Slice3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02460h

This register stores the value of the count of pixels that have passed the depth test in Slice3. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0..1	63:32	**Depth Count UDW** This register reflects the depth test in slice3 (i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.
	31:0	**Depth Count LDW** This register reflects the depth test in slice3 (i.e., will be visible). All pixels are counted when Statistics Enable is set in the Windower State. See the Windower chapter of the 3D volume for details. Pixels that pass the depth test but fail the stencil test will not be counted.

PS Invocation Count

PS_INVOCATION_COUNT - PS Invocation Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02348h

DWord	Bit	Description
0..1	63:32	**PS Invocation Count UDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.
	31:0	**PS Invocation Count LDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

PS Invocation Count for Slice0

PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	022C8h

This register stores the value of the count of pixels that get shaded in Slice0. This register is part of the context save and restore. This register should not be programmed by SW.Note: In GT3 mode there are two slices Slice0 (Upper Slice) and Slice1 (Lower slice).

Workaround
HW reports this count 4X the actual value and therefore SW must divide the count by 4 for correct reporting.

DWord	Bit	Description
0..1	63:32	**PS Invocation Count UDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice0. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.
	31:0	**PS Invocation Count LDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice0. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

PS Invocation Count for Slice1

PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	022F0h

This register stores the value of the count of pixels that get shaded in Slice0. This register is part of the context save and restore. This register should not be programmed by SW.Note: In GT3 mode there are two slices Slice0 (Upper Slice) and Slice1 (Lower slice).

Workaround

Workaround: HW reports this count 4X the actual value and therefore SW must divide the count by 4 for correct reporting.

DWord	Bit	Description
0..1	63:32	**PS Invocation Count UDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice1. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.
	31:0	**PS Invocation Count LDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice1. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

PS Invocation Count for Slice2

PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02448h

This register stores the value of the count of pixels that get shaded in Slice2. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0..1	63:32	**PS Invocation Count UDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice2. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.
	31:0	**PS Invocation Count LDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice2. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

PS Invocation Count for Slice3

PS_INVOCATION_COUNT_SLICE3 - PS Invocation Count for Slice3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02458h

This register stores the value of the count of pixels that get shaded in Slice3. This register is part of the render context save and restore. This register should not be programmed by SW.

DWord	Bit	Description
0..1	63:32	**PS Invocation Count UDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice3. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.
	31:0	**PS Invocation Count LDW** Reflects a count of the total number of pixels (including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients) that are dispatched to pixel shader invocations while Statistics Enable is set in the Windower in Slice3. See the Windower chapter of the 3D volume for details. This count will generally be much greater than the actual count of PS threads since a single thread may process up to 32 pixels.

PTBR_PAGE_POOL_OOM_EVENT_REGISTER

PTBR_PAGE_POOL_OOM_EVENT_REGISTER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PositionCS
Default Value:	0x00000000
Size (in bits):	32
Address:	1A5A4h
Name:	Register Template Address
ShortName:	ADDR_SHORT_NAME

Indicates the number of times the POSH pipe has encountered Out of Memory at any given point of time.. This register is engine context save/restored.

DWord	Bit	Description
0	31:17	**Reserved** Default Value: 000000000000000b Access: RO

PTBR Page Pool Size on Out Of Memory

PTBR_PAGE_POOL_SIZE_ON_OOM_REGISTER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PositionCS
Default Value:	0x00000000
Size (in bits):	32
Address:	18594h
Name:	Register Template Address
ShortName:	PTBR_PAGE_POOL_SIZE_ON_OOM_REGISTER

Indicates the PTBR_PAGE_POOL_SIZE when POSH pipe has encountered out of memory. This register is engine context save/restored.

DWord	Bit	Description
0	31:17	**Reserved** Default Value: 000000000000000b Access: RO

PTE SW Fault Repair High

PTESWC_H - PTE SW Fault Repair High	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04104h

DWord	Bit	Description
0	31:0	**PTE SW Fault Repair High** Default Value: 00000000h Access: R/W Fixed PTE entry is written by SW here.

PTE SW Fault Repair Low

PTESWC_L - PTE SW Fault Repair Low		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04100h	
DWord	**Bit**	**Description**
0	31:0	**PTE SW Fault Repair Low** Default Value: 00000000h Access: R/W Fixed PTE entry is written by SW here.

Punit to Gunit Message

P2GMESSAGE - Punit to Gunit Message

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300A8h

This register is used for messaging communication between Punit and Gunit.

DWord	Bit	Description
0	31:16	**P2GMSGMSK** Default Value: 0000h Access: RO Mask bits for lower 16 bits to avoid a read/modify write if '0' the corresponding bit in [15:0] is not changed if '1' the corresponding bit in [15:0] is changed to the value in [15:0]
	15	**Punit to Gunit Ack for (EU/SS) resource profile change** Default Value: 0b Access: WO New for CHV, BSW. Punit sends an ack to proceed with changing the EU/SS requested by CS. This is self clearing bit. Both this bit and bit 31 should be '1 and byte enables should be set for writing into this bit. Once written, ack will stay high for one czclk wide and self clears. Reads will always return '0. Other bits that are involved in the flow are A11C[28] and A11C[27].Sequencing. 1) CS sends (EU/SS) resource request to GPM. 2) GPM writes to offset 0xDA in Punit config space showing what the intended new configuration will be, when config bit allows. 3) Punit has opportunity to do whatever is needed, but can not wait for CPDack as it will result in a hang. 4) Based on a config unit setting, GPM will wait for an acknowledge from Punit. By default, GPM does not wait for ack due to large latency. 5) GPM continues to change the (EU/SS) resource profile. 6) GPM replies to CS with the new resource profile. 7) Execute the workload or arbitrate CPD.
	14:1	**P2G_MSG_RSVD** Default Value: 0000h Access: R/W P2GMSGReserved

P2GMESSAGE - Punit to Gunit Message

	0	**P2GMSG0** Default Value: 0b Access: R/W Context restore. this bit is used as part of the hardware context restore process. Step1: Punit writes '1(with mask bit 16 set) to initiate the HW context restore(for registers driver would normally restore) Step2: Gunit goes through context restore. Step3: Gunit writes a 1 to Punit register DB[0] indicating context restore has completed. This bit is cleared automatically when Gunit is powered down. Punit must do the same. The specific usage of this bit limits it to only be set once, for initial context restore request after an s0ix exit. Any subsequent write is undefined.

PWRCTXSAVE Message Register for Power Managment Unit

MSG_PWRCTXSAVE_GPM - PWRCTXSAVE Message Register for Power Managment Unit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08044h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.

DWord	Bit	Description
0	15:10	**Reserved** Access: RO
	9	**Power Context Save Request** Access: R/W Power Context Save Request 1'b0 : Power context save is not being requested <default> 1'b1 : Power context save is being requested Unit needs to self-clear this bit upon sampling.
	8:0	**QWord Credits for Power Context Save Request** Access: R/W QWord Credits for Power Context Save Request Minimum Credits = 1 : Unit may send 1 QWord pair (enough for first LRI at least) Maximum Credits = 511 : Unit may send 511 QWord pairs A QWord pair is defined as a 32-bit register address and the corresponding 32-bits of register data. Note that the LRI header and END commands are 64-bits each (32-bit command followed by 32-bit NOOP) and will consume one QWord credit. Only valid with PWRCTXSAVE_REQ (Bit9).

PWRMTR_WT1_EEVT1TO4

PWRMTR_WT1_EEVT1TO4 - PWRMTR_WT1_EEVT1TO4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A850h

PWRMTR_WT1_EEVT1to4

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_EU_TH_EVT4** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EU_TH_EVT4 This event measured by this field can be overwritten by AACC[3].
	23:16	**PWRMTR_WT1_EU_GA_EVT3** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EU_GA_EVT3 This event measured by this field can be overwritten by AACC[2].
	15:8	**PWRMTR_WT1_EU_GA_EVT2** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EU_GA_EVT2 This event measured by this field can be overwritten by AACC[1].
	7:0	**PWRMTR_WT1_EU_GA_EVT1** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EU_GA_EVT1 This event measured by this field can be overwritten by AACC[0].

PWRMTR_WT1_EEVT5TO8

PWRMTR_WT1_EEVT5TO8 - PWRMTR_WT1_EEVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A854h

PWRMTR_WT1_EEVT5to8

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E0_HP_EVT8** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_HP_EVT8 This event measured by this field can be overwritten by AACC[7].
	23:16	**PWRMTR_WT1_E0_HP_EVT7** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_HP_EVT7 This event measured by this field can be overwritten by AACC[6].
	15:8	**PWRMTR_WT1_E0_HP_EVT6** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_HP_EVT6 This event measured by this field can be overwritten by AACC[5].
	7:0	**PWRMTR_WT1_EU_IO_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EU_IO_EVT5 This event measured by this field can be overwritten by AACC[4].

PWRMTR_WT1_EEVT9TO12

PWRMTR_WT1_EEVT9TO12 - PWRMTR_WT1_EEVT9TO12		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A858h	
PWRMTR_WT1_EEVT9to12		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT1_E0_SP_EVT12** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_SP_EVT12 This event measured by this field can be overwritten by AACC[11].
	23:16	**PWRMTR_WT1_E0_SP_EVT11** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_SP_EVT11 This event measured by this field can be overwritten by AACC[10].
	15:8	**PWRMTR_WT1_E0_SP_EVT10** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_SP_EVT10 This event measured by this field can be overwritten by AACC[9].
	7:0	**PWRMTR_WT1_E0_HP_EVT9** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_HP_EVT9 This event measured by this field can be overwritten by AACC[8].

PWRMTR_WT1_EEVT13TO16

PWRMTR_WT1_EEVT13TO16 - PWRMTR_WT1_EEVT13TO16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A85Ch

PWRMTR_WT1_EEVT13to16

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E0_DP_EVT16** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_DP_EVT16 This event measured by this field can be overwritten by AACC[15]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT1_E0_DP_EVT15** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_DP_EVT15 This event measured by this field can be overwritten by AACC[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT1_E0_DP_EVT14** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_DP_EVT14 This event measured by this field can be overwritten by AACC[13]. **Programming Notes** Must be 0 at all times.
	7:0	**PWRMTR_WT1_E0_SP_EVT13** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_SP_EVT13 This event measured by this field can be overwritten by AACC[12].

PWRMTR_WT1_EEVT17TO20

PWRMTR_WT1_EEVT17TO20 - PWRMTR_WT1_EEVT17TO20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A860h

PWRMTR_WT1_EEVT17to20

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E0INT_EVT20** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0INT_EVT20 This event measured by this field can be overwritten by AACC[19].
	23:16	**PWRMTR_WT1_E0INT_EVT19** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0INT_EVT19 This event measured by this field can be overwritten by AACC[18]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT1_E0INT_EVT18** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0INT_EVT18 This event measured by this field can be overwritten by AACC[17].
	7:0	**PWRMTR_WT1_E0_DP_EVT17** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_DP_EVT17 This event measured by this field can be overwritten by AACC[16]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT1_EEVT21TO24

PWRMTR_WT1_EEVT21TO24 - PWRMTR_WT1_EEVT21TO24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A864h

PWRMTR_WT1_EEVT21to24

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E0_QP_EVT24** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_QP_EVT24 This event measured by this field can be overwritten by AACC[23]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT1_E0_QP_EVT23** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_QP_EVT23 This event measured by this field can be overwritten by AACC[22]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT1_E0_QP_EVT22** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_QP_EVT22 This event measured by this field can be overwritten by AACC[21].
	7:0	**PWRMTR_WT1_E0INT_EVT21** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0INT_EVT21 This event measured by this field can be overwritten by AACC[20].

PWRMTR_WT1_EEVT25TO28

PWRMTR_WT1_EEVT25TO28 - PWRMTR_WT1_EEVT25TO28	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A868h

PWRMTR_WT1_EEVT25to28

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_EFPU0_EVT28** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU0_EVT28 This event measured by this field can be overwritten by AACC[27].
	23:16	**PWRMTR_WT1_EFPU0_EVT27** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU0_EVT27 This event measured by this field can be overwritten by AACC[26].
	15:8	**PWRMTR_WT1_EFPU0_EVT26** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU0_EVT26 This event measured by this field can be overwritten by AACC[25].
	7:0	**PWRMTR_WT1_E0_QP_EVT25** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E0_QP_EVT25 This event measured by this field can be overwritten by AACC[24]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT1_EEVT29TO32

PWRMTR_WT1_EEVT29TO32 - PWRMTR_WT1_EEVT29TO32

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A86Ch

PWRMTR_WT1_EEVT29to32

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E1_HP_EVT32** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_HP_EVT32 This event measured by this field can be overwritten by AACC[31].
	23:16	**PWRMTR_WT1_E1_HP_EVT31** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_HP_EVT31 This event measured by this field can be overwritten by AACC[30].
	15:8	**PWRMTR_WT1_E1_HP_EVT30** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_HP_EVT30 This event measured by this field can be overwritten by AACC[29].
	7:0	**PWRMTR_WT1_EFPU0_EVT29** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU0_EVT29 This event measured by this field can be overwritten by AACC[28].

PWRMTR_WT1_EEVT33TO36

PWRMTR_WT1_EEVT33TO36 - PWRMTR_WT1_EEVT33TO36	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A870h

PWRMTR_WT1_EEVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E1_SP_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_SP_EVT36 This event measured by this field can be overwritten by AAD0[3].
	23:16	**PWRMTR_WT1_E1_SP_EVT35** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_SP_EVT35 This event measured by this field can be overwritten by AAD0[2].
	15:8	**PWRMTR_WT1_E1_SP_EVT34** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_SP_EVT34 This event measured by this field can be overwritten by AAD0[1].
	7:0	**PWRMTR_WT1_E1_HP_EVT33** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_HP_EVT33 This event measured by this field can be overwritten by AAD0[0].

PWRMTR_WT1_EEVT37TO40

PWRMTR_WT1_EEVT37TO40 - PWRMTR_WT1_EEVT37TO40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A874h

PWRMTR_WT1_EEVT37to40

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E1_DP_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_DP_EVT40 This event measured by this field can be overwritten by AAD0[7].
	23:16	**PWRMTR_WT1_E1_DP_EVT39** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_DP_EVT39 This event measured by this field can be overwritten by AAD0[6].
	15:8	**PWRMTR_WT1_E1_DP_EVT38** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_DP_EVT38 This event measured by this field can be overwritten by AAD0[5].
	7:0	**PWRMTR_WT1_E1_SP_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_SP_EVT37 This event measured by this field can be overwritten by AAD0[4].

PWRMTR_WT1_EEVT41TO44

PWRMTR_WT1_EEVT41TO44 - PWRMTR_WT1_EEVT41TO44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A878h

PWRMTR_WT1_EEVT41to44

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E1INT_EVT44** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1INT_EVT44 This event measured by this field can be overwritten by AAD0[11].
	23:16	**PWRMTR_WT1_E1INT_EVT43** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1INT_EVT43 This event measured by this field can be overwritten by AAD0[10]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT1_E1INT_EVT42** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1INT_EVT42 This event measured by this field can be overwritten by AAD0[9].
	7:0	**PWRMTR_WT1_E1_DP_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_DP_EVT41 This event measured by this field can be overwritten by AAD0[8].

PWRMTR_WT1_EEVT45TO48

PWRMTR_WT1_EEVT45TO48 - PWRMTR_WT1_EEVT45TO48

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A87Ch

PWRMTR_WT1_EEVT45to48

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_E1_QP_EVT48** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_QP_EVT48 This event measured by this field can be overwritten by AAD0[15].
	23:16	**PWRMTR_WT1_E1_QP_EVT47** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_QP_EVT47 This event measured by this field can be overwritten by AAD0[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT1_E1_QP_EVT46** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_QP_EVT46 This event measured by this field can be overwritten by AAD0[13].
	7:0	**PWRMTR_WT1_E1INT_EVT45** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1INT_EVT45 This event measured by this field can be overwritten by AAD0[12].

PWRMTR_WT1_EEVT49TO52

PWRMTR_WT1_EEVT49TO52 - PWRMTR_WT1_EEVT49TO52		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A880h	
PWRMTR_WT1_EEVT49to52		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT1_EFPU1_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU1_EVT52 This event measured by this field can be overwritten by AAD0[19].
	23:16	**PWRMTR_WT1_EFPU1_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU1_EVT51 This event measured by this field can be overwritten by AAD0[18].
	15:8	**PWRMTR_WT1_EFPU1_EVT50** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU1_EVT50 This event measured by this field can be overwritten by AAD0[17].
	7:0	**PWRMTR_WT1_E1_QP_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT1_E1_QP_EVT49 This event measured by this field can be overwritten by AAD0[16].

PWRMTR_WT1_EEVT53TO56

PWRMTR_WT1_EEVT53TO56 - PWRMTR_WT1_EEVT53TO56

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A884h

PWRMTR_WT1_EEVT53to56

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_EEMEM_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EEMEM_EVT56 This event measured by this field can be overwritten by AAD0[23].
	23:16	**PWRMTR_WT1_EEMEM_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EEMEM_EVT55 This event measured by this field can be overwritten by AAD0[22].
	15:8	**PWRMTR_WT1_EEMEM_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EEMEM_EVT54 This event measured by this field can be overwritten by AAD0[21].
	7:0	**PWRMTR_WT1_EFPU1_EVT53** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EFPU1_EVT53 This event measured by this field can be overwritten by AAD0[20].

PWRMTR_WT1_EEVT57TO58

PWRMTR_WT1_EEVT57TO58 - PWRMTR_WT1_EEVT57TO58	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A888h
PWRMTR_WT1_EEVT57to58	

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT1_EBUSS_EVT58** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EBUSS_EVT58 This event measured by this field can be overwritten by AAD0[25].
	7:0	**PWRMTR_WT1_EEMEM_EVT57** Default Value: 00h Access: R/W Lock PWRMTR_WT1_EEMEM_EVT57 This event measured by this field can be overwritten by AAD0[24].

PWRMTR_WT1_MEVT1TO4

PWRMTR_WT1_MEVT1TO4 - PWRMTR_WT1_MEVT1TO4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A890h

PWRMTR_WT1_MEVT1to4

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDOI_EVT4** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOI_EVT4 This event measured by this field can be overwritten by AAD4[3].
	23:16	**PWRMTR_WT1_MMSOI_EVT3** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMSOI_EVT3 This event measured by this field can be overwritten by AAD4[2].
	15:8	**PWRMTR_WT1_MMSOI_EVT2** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMSOI_EVT2 This event measured by this field can be overwritten by AAD4[1].
	7:0	**PWRMTR_WT1_MMSOI_EVT1** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMSOI_EVT1 This event measured by this field can be overwritten by AAD4[0].

PWRMTR_WT1_MEVT5TO8

PWRMTR_WT1_MEVT5TO8 - PWRMTR_WT1_MEVT5TO8	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A894h
PWRMTR_WT1_MEVT5to8	

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDOI_EVT8** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOI_EVT8 This event measured by this field can be overwritten by AAD4[7].
	23:16	**PWRMTR_WT1_MMDOI_EVT7** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOI_EVT7 This event measured by this field can be overwritten by AAD4[6].
	15:8	**PWRMTR_WT1_MMDOI_EVT6** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOI_EVT6 This event measured by this field can be overwritten by AAD4[5].
	7:0	**PWRMTR_WT1_MMDOI_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOI_EVT5 This event measured by this field can be overwritten by AAD4[4].

PWRMTR_WT1_MEVT9TO12

PWRMTR_WT1_MEVT9TO12 - PWRMTR_WT1_MEVT9TO12

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A898h

PWRMTR_WT1_MEVT9to12

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDCN_EVT12** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT12 This event measured by this field can be overwritten by AAD4[11].
	23:16	**PWRMTR_WT1_MMDCN_EVT11** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT11 This event measured by this field can be overwritten by AAD4[10].
	15:8	**PWRMTR_WT1_MMDCN_EVT10** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT10 This event measured by this field can be overwritten by AAD4[9].
	7:0	**PWRMTR_WT1_MMDCN_EVT9** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT9 This event measured by this field can be overwritten by AAD4[8].

PWRMTR_WT1_MEVT13TO16

PWRMTR_WT1_MEVT13TO16 - PWRMTR_WT1_MEVT13TO16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A89Ch

PWRMTR_WT1_MEVT13to16

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDCN_EVT16** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT16 This event measured by this field can be overwritten by AAD4[15].
	23:16	**PWRMTR_WT1_MMDCN_EVT15** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT15 This event measured by this field can be overwritten by AAD4[14].
	15:8	**PWRMTR_WT1_MMDCN_EVT14** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT14 This event measured by this field can be overwritten by AAD4[13].
	7:0	**PWRMTR_WT1_MMDCN_EVT13** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT13 This event measured by this field can be overwritten by AAD4[12].

PWRMTR_WT1_MEVT17TO20

PWRMTR_WT1_MEVT17TO20 - PWRMTR_WT1_MEVT17TO20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A8A0h

PWRMTR_WT1_MEVT17to20

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDMH_EVT20** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDMH_EVT20 This event measured by this field can be overwritten by AAD4[19].
	23:16	**PWRMTR_WT1_MMDCN_EVT19** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT19 This event measured by this field can be overwritten by AAD4[18].
	15:8	**PWRMTR_WT1_MMDCN_EVT18** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT18 This event measured by this field can be overwritten by AAD4[17].
	7:0	**PWRMTR_WT1_MMDCN_EVT17** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT17 This event measured by this field can be overwritten by AAD4[16].

PWRMTR_WT1_MEVT21TO24

PWRMTR_WT1_MEVT21TO24 - PWRMTR_WT1_MEVT21TO24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A8A4h

PWRMTR_WT1_MEVT21to24

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDOH_EVT24** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOH_EVT24 This event measured by this field can be overwritten by AAD4[23].
	23:16	**PWRMTR_WT1_MMDMH_EVT23** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDMH_EVT23 This event measured by this field can be overwritten by AAD4[22].
	15:8	**PWRMTR_WT1_MMDMH_EVT22** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDMH_EVT22 This event measured by this field can be overwritten by AAD4[21].
	7:0	**PWRMTR_WT1_MMDMH_EVT21** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDMH_EVT21 This event measured by this field can be overwritten by AAD4[20].

PWRMTR_WT1_MEVT25TO28

PWRMTR_WT1_MEVT25TO28 - PWRMTR_WT1_MEVT25TO28

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A8A8h

PWRMTR_WT1_MEVT25to28

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MMDCN_EVT28** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT28 This event measured by this field can be overwritten by AAD4[27].
	23:16	**PWRMTR_WT1_MMDCN_EVT27** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT27 This event measured by this field can be overwritten by AAD4[26].
	15:8	**PWRMTR_WT1_MMDCN_EVT26** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT26 This event measured by this field can be overwritten by AAD4[25]. Programming Notes: Must be 0 at all times. — Project: CHV, BSW
	7:0	**PWRMTR_WT1_MMDOH_EVT25** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDOH_EVT25 This event measured by this field can be overwritten by AAD4[24].

PWRMTR_WT1_MEVT29TO32

PWRMTR_WT1_MEVT29TO32 - PWRMTR_WT1_MEVT29TO32	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A8ACh

PWRMTR_WT1_MEVT29to32

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MHECN_EVT32** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT32 This event measured by this field can be overwritten by AAD4[31].
	23:16	**PWRMTR_WT1_MHECN_EVT31** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT31 This event measured by this field can be overwritten by AAD4[30].
	15:8	**PWRMTR_WT1_MHECN_EVT30** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT30 This event measured by this field can be overwritten by AAD4[29].
	7:0	**PWRMTR_WT1_MMDCN_EVT29** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MMDCN_EVT29 This event measured by this field can be overwritten by AAD4[28].

PWRMTR_WT1_MEVT33TO36

PWRMTR_WT1_MEVT33TO36 - PWRMTR_WT1_MEVT33TO36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A8B0h

PWRMTR_WT1_MEVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_MHECN_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT36 This event measured by this field can be overwritten by AAD8[3].
	23:16	**PWRMTR_WT1_MHECN_EVT35** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT35 This event measured by this field can be overwritten by AAD8[2].
	15:8	**PWRMTR_WT1_MHECN_EVT34** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT34 This event measured by this field can be overwritten by AAD8[1].
	7:0	**PWRMTR_WT1_MHECN_EVT33** Default Value: 00h Access: R/W Lock PWRMTR_WT1_MHECN_EVT33 This event measured by this field can be overwritten by AAD8[0].

PWRMTR_WT1_REVT1TO4

PWRMTR_WT1_REVT1TO4 - PWRMTR_WT1_REVT1TO4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A800h

PWRMTR_WT1_REVT1to4

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RFFOI_EVT4** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT4. This event measured by this field can be overwritten by AAC0[3].
	23:16	**PWRMTR_WT1_RFFOI_EVT3** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT3. This event measured by this field can be overwritten by AAC0[2].
	15:8	**PWRMTR_WT1_RFFOI_EVT2** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT2. This event measured by this field can be overwritten by AAC0[1].
	7:0	**PWRMTR_WT1_RFFOI_EVT1** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT1.This event measured by this field can be overwritten by AAC0[0].

PWRMTR_WT1_REVT5TO8

PWRMTR_WT1_REVT5TO8 - PWRMTR_WT1_REVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A804h

PWRMTR_WT1_REVT5to8

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RFFOI_EVT8** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT8. This event measured by this field can be overwritten by AAC0[7].
	23:16	**PWRMTR_WT1_RFFOI_EVT7** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT7. This event measured by this field can be overwritten by AAC0[6].
	15:8	**PWRMTR_WT1_RFFOI_EVT6** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT6.This event measured by this field can be overwritten by AAC0[5].
	7:0	**PWRMTR_WT1_RFFOI_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT5.This event measured by this field can be overwritten by AAC0[4].

PWRMTR_WT1_REVT9TO12

PWRMTR_WT1_REVT9TO12 - PWRMTR_WT1_REVT9TO12		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A808h	
PWRMTR_WT1_REVT9to12		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT1_RFFOI_EVT12** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT12.This event measured by this field can be overwritten by AAC0[11].
	23:16	**PWRMTR_WT1_RFFOI_EVT11** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT11.This event measured by this field can be overwritten by AAC0[10].
	15:8	**PWRMTR_WT1_RFFOI_EVT10** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT10. This event measured by this field can be overwritten by AAC0[9].
	7:0	**PWRMTR_WT1_RFFOI_EVT9** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT9.This event measured by this field can be overwritten by AAC0[8].

PWRMTR_WT1_REVT13TO16

PWRMTR_WT1_REVT13TO16 - PWRMTR_WT1_REVT13TO16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A80Ch

PWRMTR_WT1_REVT13to16

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RFFCN_EVT16** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT16.This event measured by this field can be overwritten by AAC0[15]. Programming Notes: Must be 0 at all times. Project: CHV, BSW
	23:16	**PWRMTR_WT1_RFFCN_EVT15** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT15. This event measured by this field can be overwritten by AAC0[14].
	15:8	**PWRMTR_WT1_RFFOI_EVT14** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT14.This event measured by this field can be overwritten by AAC0[13].
	7:0	**PWRMTR_WT1_RFFOI_EVT13** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFOI_EVT13.This event measured by this field can be overwritten by AAC0[12].

PWRMTR_WT1_REVT17TO20

PWRMTR_WT1_REVT17TO20 - PWRMTR_WT1_REVT17TO20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A810h

PWRMTR_WT1_REVT17to20

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RFFCN_EVT20** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT20.Thiseventmeasuredbythisfieldcanbeoverwrittenby AAC0[19].
	23:16	**PWRMTR_WT1_RFFCN_EVT19** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT19 Thiseventmeasuredbythisfieldcanbeoverwrittenby AAC0[18].
	15:8	**PWRMTR_WT1_RFFCN_EVT18** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT18 This event measured by this field can be overwritten by AAC0[17].
	7:0	**PWRMTR_WT1_RFFCN_EVT17** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RFFCN_EVT17 This event measured by this field can be overwritten by AAC0[16]. Programming Notes: Must be 0 at all times. — Project: CHV, BSW

PWRMTR_WT1_REVT21TO24

PWRMTR_WT1_REVT21TO24 - PWRMTR_WT1_REVT21TO24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A814h

PWRMTR_WT1_REVT21to24

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RSCOI_EVT24** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT24 This event measured by this field can be overwritten by AAC0[23].
	23:16	**PWRMTR_WT1_RSCOI_EVT23** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT23 This event measured by this field can be overwritten by AAC0[22].
	15:8	**PWRMTR_WT1_RSCOI_EVT22** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT22 This event measured by this field can be overwritten by AAC0[21].
	7:0	**PWRMTR_WT1_RSCOI_EVT21** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT21 This event measured by this field can be overwritten by AAC0[20].

PWRMTR_WT1_REVT25TO28

PWRMTR_WT1_REVT25TO28 - PWRMTR_WT1_REVT25TO28	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A818h
PWRMTR_WT1_REVT25to28	

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RSCOI_EVT28** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT28 This event measured by this field can be overwritten by AAC0[27].
	23:16	**PWRMTR_WT1_RSCOI_EVT27** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT27 This event measured by this field can be overwritten by AAC0[26].
	15:8	**PWRMTR_WT1_RSCOI_EVT26** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT26 This event measured by this field can be overwritten by AAC0[25].
	7:0	**PWRMTR_WT1_RSCOI_EVT25** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT25 This event measured by this field can be overwritten by AAC0[24].

PWRMTR_WT1_REVT29TO32

PWRMTR_WT1_REVT29TO32 - PWRMTR_WT1_REVT29TO32

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A81Ch

PWRMTR_WT1_REVT29to32

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RSCOI_EVT32** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT32 This event measured by this field can be overwritten by AAC0[31].
	23:16	**PWRMTR_WT1_RSCOI_EVT31** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT31 This event measured by this field can be overwritten by AAC0[30].
	15:8	**PWRMTR_WT1_RSCOI_EVT30** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT30 This event measured by this field can be overwritten by AAC0[29].
	7:0	**PWRMTR_WT1_RSCOI_EVT29** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT29 This event measured by this field can be overwritten by AAC0[28].

PWRMTR_WT1_REVT33TO36

PWRMTR_WT1_REVT33TO36 - PWRMTR_WT1_REVT33TO36	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A820h
PWRMTR_WT1_REVT33to36	

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RL3MI_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RL3MI_EVT36 This event measured by this field can be overwritten by AAC4[3].
	23:16	**PWRMTR_WT1_RL3CN_EVT35** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RL3CN_EVT35 This event measured by this field can be overwritten by AAC4[2].
	15:8	**PWRMTR_WT1_RSCCN_EVT34** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCCN_EVT34 This event measured by this field can be overwritten by AAC4[1].
	7:0	**PWRMTR_WT1_RSCOI_EVT33** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RSCOI_EVT33 This event measured by this field can be overwritten by AAC4[0].

PWRMTR_WT1_REVT37TO40

PWRMTR_WT1_REVT37TO40 - PWRMTR_WT1_REVT37TO40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A824h

PWRMTR_WT1_REVT37to40

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTAOI_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT40 This event measured by this field can be overwritten by AAC4[7].
	23:16	**PWRMTR_WT1_RL3MI_EVT39** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RL3MI_EVT39 This event measured by this field can be overwritten by AAC4[6].
	15:8	**PWRMTR_WT1_RL3MI_EVT38** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RL3MI_EVT38 This event measured by this field can be overwritten by AAC4[5].
	7:0	**PWRMTR_WT1_RL3MI_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RL3MI_EVT37 This event measured by this field can be overwritten by AAC4[4].

PWRMTR_WT1_REVT41TO44

PWRMTR_WT1_REVT41TO44 - PWRMTR_WT1_REVT41TO44		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A828h	
PWRMTR_WT1_REVT41to44		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT1_RTAOI_EVT44** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT44 This event measured by this field can be overwritten by AAC4[11].
	23:16	**PWRMTR_WT1_RTAOI_EVT43** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT43 This event measured by this field can be overwritten by AAC4[10].
	15:8	**PWRMTR_WT1_RTAOI_EVT42** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT42 This event measured by this field can be overwritten by AAC4[9].
	7:0	**PWRMTR_WT1_RTAOI_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT41 This event measured by this field can be overwritten by AAC4[8].

PWRMTR_WT1_REVT45TO48

PWRMTR_WT1_REVT45TO48 - PWRMTR_WT1_REVT45TO48

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A82Ch

PWRMTR_WT1_REVT45to48

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTAOI_EVT48** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT48 This event measured by this field can be overwritten by AAC4[15].
	23:16	**PWRMTR_WT1_RTAOI_EVT47** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT47 This event measured by this field can be overwritten by AAC4[14].
	15:8	**PWRMTR_WT1_RTAOI_EVT46** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT46 This event measured by this field can be overwritten by AAC4[13].
	7:0	**PWRMTR_WT1_RTAOI_EVT45** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT45 This event measured by this field can be overwritten by AAC4[12].

PWRMTR_WT1_REVT49TO52

PWRMTR_WT1_REVT49TO52 - PWRMTR_WT1_REVT49TO52	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A830h
PWRMTR_WT1_REVT49to52	

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTAOH_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOH_EVT52 This event measured by this field can be overwritten by AAC4[19].
	23:16	**PWRMTR_WT1_RTAOH_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOH_EVT51 This event measured by this field can be overwritten by AAC4[18].
	15:8	**PWRMTR_WT1_RTAOI_EVT50** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT50 This event measured by this field can be overwritten by AAC4[17].
	7:0	**PWRMTR_WT1_RTAOI_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTAOI_EVT49 This event measured by this field can be overwritten by AAC4[16].

PWRMTR_WT1_REVT53TO56

PWRMTR_WT1_REVT53TO56 - PWRMTR_WT1_REVT53TO56

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A834h

PWRMTR_WT1_REVT53to56

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTSOI_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTS OI_EVT56 This event measured by this field can be overwritten by AAC4[23].
	23:16	**PWRMTR_WT1_RTACN_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTACN_EVT55 This event measured by this field can be overwritten by AAC4[22].
	15:8	**PWRMTR_WT1_RTACN_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTACN_EVT54 This event measured by this field can be overwritten by AAC4[21].
	7:0	**PWRMTR_WT1_RTACN_EVT53** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTACN_EVT53 This event measured by this field can be overwritten by AAC4[20].

PWRMTR_WT1_REVT57TO60

PWRMTR_WT1_REVT57TO60 - PWRMTR_WT1_REVT57TO60

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A838h

PWRMTR_WT1_REVT57to60

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTSOI_EVT60** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTS OI_EVT60 This event measured by this field can be overwritten by AAC4[27].
	23:16	**PWRMTR_WT1_RTSOI_EVT59** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTS OI_EVT59 This event measured by this field can be overwritten by AAC4[26].
	15:8	**PWRMTR_WT1_RTSOI_EVT58** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTS OI_EVT58 This event measured by this field can be overwritten by AAC4[25]. **Programming Notes** Must be set to zero at all times.
	7:0	**PWRMTR_WT1_RTSOI_EVT57** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTS OI_EVT57 This event measured by this field can be overwritten by AAC4[24]. **Programming Notes** Must be set to zero at all times.

PWRMTR_WT1_REVT61TO64

PWRMTR_WT1_REVT61TO64 - PWRMTR_WT1_REVT61TO64

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A83Ch

PWRMTR_WT1_REVT61to64

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RTOOI_EVT64** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTOOI_EVT64 This event measured by this field can be overwritten by AAC4[31].
	23:16	**PWRMTR_WT1_RTOOI_EVT63** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTOOI_EVT63 This event measured by this field can be overwritten by AAC4[30].
	15:8	**PWRMTR_WT1_RTOOI_EVT62** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTOOI_EVT62 This event measured by this field can be overwritten by AAC4[29].
	7:0	**PWRMTR_WT1_RTOOI_EVT61** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RTOOI_EVT61 This event measured by this field can be overwritten by AAC4[28].

PWRMTR_WT1_REVT65TO68

PWRMTR_WT1_REVT65TO68 - PWRMTR_WT1_REVT65TO68

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A840h

PWRMTR_WT1_REVT65to68

DWord	Bit	Description
0	31:24	**PWRMTR_WT1_RGTCN_EVT68** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTCN_EVT68 This event measured by this field can be overwritten by AAC8[3].
	23:16	**PWRMTR_WT1_RGTCN_EVT67** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTCN_EVT67 This event measured by this field can be overwritten by AAC8[2].
	15:8	**PWRMTR_WT1_RGTCN_EVT66** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTCN_EVT66 This event measured by this field can be overwritten by AAC8[1].
	7:0	**PWRMTR_WT1_RGTOI_EVT65** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTOI_EVT65 This event measured by this field can be overwritten by AAC8[0].

PWRMTR_WT1_REVT69TO70

PWRMTR_WT1_REVT69TO70 - PWRMTR_WT1_REVT69TO70

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A844h

PWRMTR_WT1_REVT69to70

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT1_RGTCN_EVT70** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTCN_EVT70 This event measured by this field can be overwritten by AAC8[5].
	7:0	**PWRMTR_WT1_RGTCN_EVT69** Default Value: 00h Access: R/W Lock PWRMTR_WT1_RGTCN_EVT69 This event measured by this field can be overwritten by AAC8[4].

PWRMTR_WT2_EEVT1TO4

PWRMTR_WT2_EEVT1TO4 - PWRMTR_WT2_EEVT1TO4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A950h

PWRMTR_WT2_EEVT1to4

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_EU_TH_EVT4** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EU_TH_EVT4 This event measured by this field can be overwritten by AACC[3].
	23:16	**PWRMTR_WT2_EU_GA_EVT3** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EU_GA_EVT3 This event measured by this field can be overwritten by AACC[2].
	15:8	**PWRMTR_WT2_EU_GA_EVT2** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EU_GA_EVT2 This event measured by this field can be overwritten by AACC[1].
	7:0	**PWRMTR_WT2_EU_GA_EVT1** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EU_GA_EVT1 This event measured by this field can be overwritten by AACC[0].

PWRMTR_WT2_EEVT5TO8

PWRMTR_WT2_EEVT5TO8 - PWRMTR_WT2_EEVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A954h

PWRMTR_WT2_EEVT5to8

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E0_HP_EVT8** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_HP_EVT8 This event measured by this field can be overwritten by AACC[7].
	23:16	**PWRMTR_WT2_E0_HP_EVT7** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_HP_EVT7 This event measured by this field can be overwritten by AACC[6].
	15:8	**PWRMTR_WT2_E0_HP_EVT6** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_HP_EVT6 This event measured by this field can be overwritten by AACC[5].
	7:0	**PWRMTR_WT2_EU_IO_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EU_IO_EVT5 This event measured by this field can be overwritten by AACC[4].

PWRMTR_WT2_EEVT9TO12

PWRMTR_WT2_EEVT9TO12 - PWRMTR_WT2_EEVT9TO12

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A958h

PWRMTR_WT2_EEVT9to12

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E0_SP_EVT12** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_SP_EVT12 This event measured by this field can be overwritten by AACC[11].
	23:16	**PWRMTR_WT2_E0_SP_EVT11** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_SP_EVT11 This event measured by this field can be overwritten by AACC[10].
	15:8	**PWRMTR_WT2_E0_SP_EVT10** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_SP_EVT10 This event measured by this field can be overwritten by AACC[9].
	7:0	**PWRMTR_WT2_E0_HP_EVT9** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_HP_EVT9 This event measured by this field can be overwritten by AACC[8].

PWRMTR_WT2_EEVT13TO16

PWRMTR_WT2_EEVT13TO16 - PWRMTR_WT2_EEVT13TO16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A95Ch

PWRMTR_WT2_EEVT13to16

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E0_DP_EVT16** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_DP_EVT16 This event measured by this field can be overwritten by AACC[15]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT2_E0_DP_EVT15** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_DP_EVT15 This event measured by this field can be overwritten by AACC[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT2_E0_DP_EVT14** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_DP_EVT14 This event measured by this field can be overwritten by AACC[13]. **Programming Notes** Must be 0 at all times.
	7:0	**PWRMTR_WT2_E0_SP_EVT13** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_SP_EVT13 This event measured by this field can be overwritten by AACC[12].

PWRMTR_WT2_EEVT17TO20

PWRMTR_WT2_EEVT17TO20 - PWRMTR_WT2_EEVT17TO20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A960h

PWRMTR_WT2_EEVT17to20

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E0INT_EVT20** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0INT_EVT20 This event measured by this field can be overwritten by AACC[19].
	23:16	**PWRMTR_WT2_E0INT_EVT19** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0INT_EVT19 This event measured by this field can be overwritten by AACC[18]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT2_E0INT_EVT18** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0INT_EVT18 This event measured by this field can be overwritten by AACC[17].
	7:0	**PWRMTR_WT2_E0_DP_EVT17** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_DP_EVT17 This event measured by this field can be overwritten by AACC[16]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT2_EEVT21TO24

PWRMTR_WT2_EEVT21TO24 - PWRMTR_WT2_EEVT21TO24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A964h

PWRMTR_WT2_EEVT21to24

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E0_QP_EVT24** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_QP_EVT24 This event measured by this field can be overwritten by AACC[23]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT2_E0_QP_EVT23** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_QP_EVT23 This event measured by this field can be overwritten by AACC[22]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT2_E0_QP_EVT22** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_QP_EVT22 This event measured by this field can be overwritten by AACC[21].
	7:0	**PWRMTR_WT2_E0INT_EVT21** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0INT_EVT21 This event measured by this field can be overwritten by AACC[20].

PWRMTR_WT2_EEVT25TO28

PWRMTR_WT2_EEVT25TO28 - PWRMTR_WT2_EEVT25TO28

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A968h

PWRMTR_WT2_EEVT25to28

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_EFPU0_EVT28** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU0_EVT28 This event measured by this field can be overwritten by AACC[27].
	23:16	**PWRMTR_WT2_EFPU0_EVT27** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU0_EVT27 This event measured by this field can be overwritten by AACC[26].
	15:8	**PWRMTR_WT2_EFPU0_EVT26** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU0_EVT26 This event measured by this field can be overwritten by AACC[25].
	7:0	**PWRMTR_WT2_E0_QP_EVT25** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E0_QP_EVT25 This event measured by this field can be overwritten by AACC[24]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT2_EEVT29TO32

PWRMTR_WT2_EEVT29TO32 - PWRMTR_WT2_EEVT29TO32

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A96Ch

PWRMTR_WT2_EEVT29to32

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E1_HP_EVT32** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_HP_EVT32 This event measured by this field can be overwritten by AACC[31].
	23:16	**PWRMTR_WT2_E1_HP_EVT31** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_HP_EVT31 This event measured by this field can be overwritten by AACC[30].
	15:8	**PWRMTR_WT2_E1_HP_EVT30** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_HP_EVT30 This event measured by this field can be overwritten by AACC[29].
	7:0	**PWRMTR_WT2_EFPU0_EVT29** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU0_EVT29 This event measured by this field can be overwritten by AACC[28].

PWRMTR_WT2_EEVT33TO36

PWRMTR_WT2_EEVT33TO36 - PWRMTR_WT2_EEVT33TO36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A970h

PWRMTR_WT2_EEVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E1_SP_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_SP_EVT36 This event measured by this field can be overwritten by AAD0[3].
	23:16	**PWRMTR_WT2_E1_SP_EVT35** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_SP_EVT35 This event measured by this field can be overwritten by AAD0[2].
	15:8	**PWRMTR_WT2_E1_SP_EVT34** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_SP_EVT34 This event measured by this field can be overwritten by AAD0[1].
	7:0	**PWRMTR_WT2_E1_HP_EVT33** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_HP_EVT33 This event measured by this field can be overwritten by AAD0[0].

PWRMTR_WT2_EEVT37TO40

PWRMTR_WT2_EEVT37TO40 - PWRMTR_WT2_EEVT37TO40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A974h

PWRMTR_WT2_EEVT37to40

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E1_DP_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_DP_EVT40 This event measured by this field can be overwritten by AAD0[7].
	23:16	**PWRMTR_WT2_E1_DP_EVT39** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_DP_EVT39 This event measured by this field can be overwritten by AAD0[6].
	15:8	**PWRMTR_WT2_E1_DP_EVT38** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_DP_EVT38 This event measured by this field can be overwritten by AAD0[5].
	7:0	**PWRMTR_WT2_E1_SP_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_SP_EVT37 This event measured by this field can be overwritten by AAD0[4].

PWRMTR_WT2_EEVT41TO44

PWRMTR_WT2_EEVT41TO44 - PWRMTR_WT2_EEVT41TO44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A978h

PWRMTR_WT2_EEVT41to44

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E1INT_EVT44** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1INT_EVT44 This event measured by this field can be overwritten by AAD0[11].
	23:16	**PWRMTR_WT2_E1INT_EVT43** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1INT_EVT43 This event measured by this field can be overwritten by AAD0[10]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT2_E1INT_EVT42** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1INT_EVT42 This event measured by this field can be overwritten by AAD0[9].
	7:0	**PWRMTR_WT2_E1_DP_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_DP_EVT41 This event measured by this field can be overwritten by AAD0[8].

PWRMTR_WT2_EEVT45TO48

PWRMTR_WT2_EEVT45TO48 - PWRMTR_WT2_EEVT45TO48

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A97Ch

PWRMTR_WT2_EEVT45to48

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_E1_QP_EVT48** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_QP_EVT48 This event measured by this field can be overwritten by AAD0[15].
	23:16	**PWRMTR_WT2_E1_QP_EVT47** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_QP_EVT47 This event measured by this field can be overwritten by AAD0[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT2_E1_QP_EVT46** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_QP_EVT46 This event measured by this field can be overwritten by AAD0[13].
	7:0	**PWRMTR_WT2_E1INT_EVT45** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1INT_EVT45 This event measured by this field can be overwritten by AAD0[12].

PWRMTR_WT2_EEVT49TO52

PWRMTR_WT2_EEVT49TO52 - PWRMTR_WT2_EEVT49TO52		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A980h	
PWRMTR_WT2_EEVT49to52		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT2_EFPU1_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU1_EVT52 This event measured by this field can be overwritten by AAD0[19].
	23:16	**PWRMTR_WT2_EFPU1_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU1_EVT51 This event measured by this field can be overwritten by AAD0[18].
	15:8	**PWRMTR_WT2_EFPU1_EVT50** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU1_EVT50 This event measured by this field can be overwritten by AAD0[17].
	7:0	**PWRMTR_WT2_E1_QP_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT2_E1_QP_EVT49 This event measured by this field can be overwritten by AAD0[16].

PWRMTR_WT2_EEVT53TO56

PWRMTR_WT2_EEVT53TO56 - PWRMTR_WT2_EEVT53TO56

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A984h

PWRMTR_WT2_EEVT53to56

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_EEMEM_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EEMEM_EVT56 This event measured by this field can be overwritten by AAD0[23].
	23:16	**PWRMTR_WT2_EEMEM_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EEMEM_EVT55 This event measured by this field can be overwritten by AAD0[22].
	15:8	**PWRMTR_WT2_EEMEM_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EEMEM_EVT54 This event measured by this field can be overwritten by AAD0[21].
	7:0	**PWRMTR_WT2_EFPU1_EVT53** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EFPU1_EVT53 This event measured by this field can be overwritten by AAD0[20].

PWRMTR_WT2_EEVT57TO58

PWRMTR_WT2_EEVT57TO58 - PWRMTR_WT2_EEVT57TO58	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A988h

PWRMTR_WT2_EEVT57to58

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT2_EBUSS_EVT58** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EBUSS_EVT58 This event measured by this field can be overwritten by AAD0[25].
	7:0	**PWRMTR_WT2_EEMEM_EVT57** Default Value: 00h Access: R/W Lock PWRMTR_WT2_EEMEM_EVT57 This event measured by this field can be overwritten by AAD0[24].

PWRMTR_WT2_REVT5TO8

PWRMTR_WT2_REVT5TO8 - PWRMTR_WT2_REVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A904h

PWRMTR_WT2_REVT5to8

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT2_RFFOI_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RFFOI_EVT5 This event measured by this field can be overwritten by AAC0[4].

PWRMTR_WT2_REVT33TO36

PWRMTR_WT2_REVT33TO36 - PWRMTR_WT2_REVT33TO36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A920h

PWRMTR_WT2_REVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_RL3MI_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RL3MI_EVT36 This event measured by this field can be overwritten by AAC4[3].
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**Reserved** Default Value: 00h Access: RO Reserved

PWRMTR_WT2_REVT37TO40

PWRMTR_WT2_REVT37TO40 - PWRMTR_WT2_REVT37TO40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A924h

PWRMTR_WT2_REVT37to40

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_RTAOI_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTAOI_EVT40 This event measured by this field can be overwritten by AAC4[7].
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT2_RL3MI_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RL3MI_EVT37 This event measured by this field can be overwritten by AAC4[4].

PWRMTR_WT2_REVT41TO44

PWRMTR_WT2_REVT41TO44 - PWRMTR_WT2_REVT41TO44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A928h

PWRMTR_WT2_REVT41to44

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT2_RTAOI_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTAOI_EVT41 This event measured by this field can be overwritten by AAC4[8].

PWRMTR_WT2_REVT49TO52

PWRMTR_WT2_REVT49TO52 - PWRMTR_WT2_REVT49TO52

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A930h

PWRMTR_WT2_REVT49to52

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_RTAOH_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTAOH_EVT52 This event measured by this field can be overwritten by AAC4[19].
	23:16	**PWRMTR_WT2_RTAOH_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTAOH_EVT51 This event measured by this field can be overwritten by AAC4[18].
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT2_RTAOI_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTAOI_EVT49 This event measured by this field can be overwritten by AAC4[16].

PWRMTR_WT2_REVT53TO56

PWRMTR_WT2_REVT53TO56 - PWRMTR_WT2_REVT53TO56	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A934h
PWRMTR_WT2_REVT53to56	

DWord	Bit	Description
0	31:24	**PWRMTR_WT2_RTSOI_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTS OI_EVT56 This event measured by this field can be overwritten by AAC4[23].
	23:16	**PWRMTR_WT2_RTACN_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTACN_EVT55 This event measured by this field can be overwritten by AAC4[22].
	15:8	**PWRMTR_WT2_RTACN_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RTACN_EVT54 This event measured by this field can be overwritten by AAC4[21].
	7:0	**Reserved** Default Value: 00h Access: RO Reserved

PWRMTR_WT2_REVT65TO68

PWRMTR_WT2_REVT65TO68 - PWRMTR_WT2_REVT65TO68

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A940h

PWRMTR_WT2_REVT65to68

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**PWRMTR_WT2_RGTCN_EVT66** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RGTCN_EVT66 This event measured by this field can be overwritten by AAC8[1].
	7:0	**PWRMTR_WT2_RGTOI_EVT65** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RGTOI_EVT65 This event measured by this field can be overwritten by AAC8[0].

PWRMTR_WT2_REVT69TO70

PWRMTR_WT2_REVT69TO70 - PWRMTR_WT2_REVT69TO70

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A944h

PWRMTR_WT2_REVT69to70

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT2_RGTCN_EVT70** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RGTCN_EVT70 This event measured by this field can be overwritten by AAC8[5].
	7:0	**PWRMTR_WT2_RGTCN_EVT69** Default Value: 00h Access: R/W Lock PWRMTR_WT2_RGTCN_EVT69 This event measured by this field can be overwritten by AAC8[4].

PWRMTR_WT3_EEVT1TO4

PWRMTR_WT3_EEVT1TO4 - PWRMTR_WT3_EEVT1TO4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA50h

PWRMTR_WT3_EEVT1to4

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_EU_TH_EVT4** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EU_TH_EVT4 This event measured by this field can be overwritten by AACC[3].
	23:16	**PWRMTR_WT3_EU_GA_EVT3** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EU_GA_EVT3 This event measured by this field can be overwritten by AACC[2].
	15:8	**PWRMTR_WT3_EU_GA_EVT2** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EU_GA_EVT2 This event measured by this field can be overwritten by AACC[1].
	7:0	**PWRMTR_WT3_EU_GA_EVT1** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EU_GA_EVT1 This event measured by this field can be overwritten by AACC[0].

PWRMTR_WT3_EEVT5TO8

PWRMTR_WT3_EEVT5TO8 - PWRMTR_WT3_EEVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA54h

PWRMTR_WT3_EEVT5to8

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E0_HP_EVT8** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_HP_EVT8 This event measured by this field can be overwritten by AACC[7].
	23:16	**PWRMTR_WT3_E0_HP_EVT7** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_HP_EVT7 This event measured by this field can be overwritten by AACC[6].
	15:8	**PWRMTR_WT3_E0_HP_EVT6** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_HP_EVT6 This event measured by this field can be overwritten by AACC[5].
	7:0	**PWRMTR_WT3_EU_IO_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EU_IO_EVT5 This event measured by this field can be overwritten by AACC[4].

PWRMTR_WT3_EEVT9TO12

PWRMTR_WT3_EEVT9TO12 - PWRMTR_WT3_EEVT9TO12

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA58h

PWRMTR_WT3_EEVT9to12

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E0_SP_EVT12** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_SP_EVT12 This event measured by this field can be overwritten by AACC[11].
	23:16	**PWRMTR_WT3_E0_SP_EVT11** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_SP_EVT11 This event measured by this field can be overwritten by AACC[10].
	15:8	**PWRMTR_WT3_E0_SP_EVT10** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_SP_EVT10 This event measured by this field can be overwritten by AACC[9].
	7:0	**PWRMTR_WT3_E0_HP_EVT9** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_HP_EVT9 This event measured by this field can be overwritten by AACC[8].

PWRMTR_WT3_EEVT13TO16

PWRMTR_WT3_EEVT13TO16 - PWRMTR_WT3_EEVT13TO16

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA5Ch

PWRMTR_WT3_EEVT13to16

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E0_DP_EVT16** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_DP_EVT16 This event measured by this field can be overwritten by AACC[15]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT3_E0_DP_EVT15** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_DP_EVT15 This event measured by this field can be overwritten by AACC[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT3_E0_DP_EVT14** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_DP_EVT14 This event measured by this field can be overwritten by AACC[13]. **Programming Notes** Must be 0 at all times.
	7:0	**PWRMTR_WT3_E0_SP_EVT13** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_SP_EVT13 This event measured by this field can be overwritten by AACC[12].

PWRMTR_WT3_EEVT17TO20

PWRMTR_WT3_EEVT17TO20 - PWRMTR_WT3_EEVT17TO20

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA60h

PWRMTR_WT3_EEVT17to20

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E0INT_EVT20** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0INT_EVT20 This event measured by this field can be overwritten by AACC[19].
	23:16	**PWRMTR_WT3_E0INT_EVT19** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0INT_EVT19 This event measured by this field can be overwritten by AACC[18]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT3_E0INT_EVT18** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0INT_EVT18 This event measured by this field can be overwritten by AACC[17].
	7:0	**PWRMTR_WT3_E0_DP_EVT17** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_DP_EVT17 This event measured by this field can be overwritten by AACC[16]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT3_EEVT21TO24

PWRMTR_WT3_EEVT21TO24 - PWRMTR_WT3_EEVT21TO24

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA64h

PWRMTR_WT3_EEVT21to24

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E0_QP_EVT24** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_QP_EVT24 This event measured by this field can be overwritten by AACC[23]. **Programming Notes** Must be 0 at all times.
	23:16	**PWRMTR_WT3_E0_QP_EVT23** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_QP_EVT23 This event measured by this field can be overwritten by AACC[22]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT3_E0_QP_EVT22** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_QP_EVT22 This event measured by this field can be overwritten by AACC[21].
	7:0	**PWRMTR_WT3_E0INT_EVT21** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0INT_EVT21 This event measured by this field can be overwritten by AACC[20].

PWRMTR_WT3_EEVT25TO28

PWRMTR_WT3_EEVT25TO28 - PWRMTR_WT3_EEVT25TO28

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA68h

PWRMTR_WT3_EEVT25to28

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_EFPU0_EVT28** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU0_EVT28 This event measured by this field can be overwritten by AACC[27].
	23:16	**PWRMTR_WT3_EFPU0_EVT27** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU0_EVT27 This event measured by this field can be overwritten by AACC[26].
	15:8	**PWRMTR_WT3_EFPU0_EVT26** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU0_EVT26 This event measured by this field can be overwritten by AACC[25].
	7:0	**PWRMTR_WT3_E0_QP_EVT25** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E0_QP_EVT25 This event measured by this field can be overwritten by AACC[24]. **Programming Notes** Must be 0 at all times.

PWRMTR_WT3_EEVT29TO32

PWRMTR_WT3_EEVT29TO32 - PWRMTR_WT3_EEVT29TO32		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AA6Ch	
PWRMTR_WT3_EEVT29to32		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT3_E1_HP_EVT32** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_HP_EVT32 This event measured by this field can be overwritten by AACC[31].
	23:16	**PWRMTR_WT3_E1_HP_EVT31** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_HP_EVT31 This event measured by this field can be overwritten by AACC[30].
	15:8	**PWRMTR_WT3_E1_HP_EVT30** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_HP_EVT30 This event measured by this field can be overwritten by AACC[29].
	7:0	**PWRMTR_WT3_EFPU0_EVT29** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU0_EVT29 This event measured by this field can be overwritten by AACC[28].

PWRMTR_WT3_EEVT33TO36

PWRMTR_WT3_EEVT33TO36 - PWRMTR_WT3_EEVT33TO36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA70h

PWRMTR_WT3_EEVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E1_SP_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_SP_EVT36 This event measured by this field can be overwritten by AAD0[3].
	23:16	**PWRMTR_WT3_E1_SP_EVT35** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_SP_EVT35 This event measured by this field can be overwritten by AAD0[2].
	15:8	**PWRMTR_WT3_E1_SP_EVT34** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_SP_EVT34 This event measured by this field can be overwritten by AAD0[1].
	7:0	**PWRMTR_WT3_E1_HP_EVT33** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_HP_EVT33 This event measured by this field can be overwritten by AAD0[0].

PWRMTR_WT3_EEVT37TO40

PWRMTR_WT3_EEVT37TO40 - PWRMTR_WT3_EEVT37TO40		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AA74h	
PWRMTR_WT3_EEVT37to40		
DWord	**Bit**	**Description**
0	31:24	**PWRMTR_WT3_E1_DP_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_DP_EVT40 This event measured by this field can be overwritten by AAD0[7].
	23:16	**PWRMTR_WT3_E1_DP_EVT39** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_DP_EVT39 This event measured by this field can be overwritten by AAD0[6].
	15:8	**PWRMTR_WT3_E1_DP_EVT38** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_DP_EVT38 This event measured by this field can be overwritten by AAD0[5].
	7:0	**PWRMTR_WT3_E1_SP_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_SP_EVT37 This event measured by this field can be overwritten by AAD0[4].

PWRMTR_WT3_EEVT41TO44

PWRMTR_WT3_EEVT41TO44 - PWRMTR_WT3_EEVT41TO44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA78h

PWRMTR_WT3_EEVT41to44

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E1INT_EVT44** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1INT_EVT44 This event measured by this field can be overwritten by AAD0[11].
	23:16	**PWRMTR_WT3_E1INT_EVT43** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1INT_EVT43 This event measured by this field can be overwritten by AAD0[10]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT3_E1INT_EVT42** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1INT_EVT42 This event measured by this field can be overwritten by AAD0[9].
	7:0	**PWRMTR_WT3_E1_DP_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_DP_EVT41 This event measured by this field can be overwritten by AAD0[8].

PWRMTR_WT3_EEVT45TO48

PWRMTR_WT3_EEVT45TO48 - PWRMTR_WT3_EEVT45TO48

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA7Ch

PWRMTR_WT3_EEVT45to48

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_E1_QP_EVT48** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_QP_EVT48 This event measured by this field can be overwritten by AAD0[15].
	23:16	**PWRMTR_WT3_E1_QP_EVT47** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_QP_EVT47 This event measured by this field can be overwritten by AAD0[14]. **Programming Notes** Must be 0 at all times.
	15:8	**PWRMTR_WT3_E1_QP_EVT46** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_QP_EVT46 This event measured by this field can be overwritten by AAD0[13].
	7:0	**PWRMTR_WT3_E1INT_EVT45** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1INT_EVT45 This event measured by this field can be overwritten by AAD0[12].

PWRMTR_WT3_EEVT49TO52

PWRMTR_WT3_EEVT49TO52 - PWRMTR_WT3_EEVT49TO52

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA80h

PWRMTR_WT3_EEVT49to52

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_EFPU1_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU1_EVT52 This event measured by this field can be overwritten by AAD0[19].
	23:16	**PWRMTR_WT3_EFPU1_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU1_EVT51 This event measured by this field can be overwritten by AAD0[18].
	15:8	**PWRMTR_WT3_EFPU1_EVT50** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU1_EVT50 This event measured by this field can be overwritten by AAD0[17].
	7:0	**PWRMTR_WT3_E1_QP_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT3_E1_QP_EVT49 This event measured by this field can be overwritten by AAD0[16].

PWRMTR_WT3_EEVT53TO56

PWRMTR_WT3_EEVT53TO56 - PWRMTR_WT3_EEVT53TO56	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA84h
PWRMTR_WT3_EEVT53to56	

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_EEMEM_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EEMEM_EVT56 This event measured by this field can be overwritten by AAD0[23].
	23:16	**PWRMTR_WT3_EEMEM_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EEMEM_EVT55 This event measured by this field can be overwritten by AAD0[22].
	15:8	**PWRMTR_WT3_EEMEM_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EEMEM_EVT54 This event measured by this field can be overwritten by AAD0[21].
	7:0	**PWRMTR_WT3_EFPU1_EVT53** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EFPU1_EVT53 This event measured by this field can be overwritten by AAD0[20].

PWRMTR_WT3_EEVT57TO58

PWRMTR_WT3_EEVT57TO58 - PWRMTR_WT3_EEVT57TO58

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA88h

PWRMTR_WT3_EEVT57to58

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT3_EBUSS_EVT58** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EBUSS_EVT58 This event measured by this field can be overwritten by AAD0[25].
	7:0	**PWRMTR_WT3_EEMEM_EVT57** Default Value: 00h Access: R/W Lock PWRMTR_WT3_EEMEM_EVT57 This event measured by this field can be overwritten by AAD0[24].

PWRMTR_WT3_REVT5TO8

PWRMTR_WT3_REVT5TO8 - PWRMTR_WT3_REVT5TO8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA04h

PWRMTR_WT3_REVT5to8

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT3_RFFOI_EVT5** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RFFOI_EVT5 This event measured by this field can be overwritten by AAC0[4].

PWRMTR_WT3_REVT33TO36

PWRMTR_WT3_REVT33TO36 - PWRMTR_WT3_REVT33TO36

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA20h

PWRMTR_WT3_REVT33to36

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_RL3MI_EVT36** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RL3MI_EVT36 This event measured by this field can be overwritten by AAC4[3].
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**Reserved** Default Value: 00h Access: RO Reserved

PWRMTR_WT3_REVT37TO40

PWRMTR_WT3_REVT37TO40 - PWRMTR_WT3_REVT37TO40

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA24h

PWRMTR_WT3_REVT37to40

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_RTAOI_EVT40** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTAOI_EVT40 This event measured by this field can be overwritten by AAC4[7].
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT3_RL3MI_EVT37** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RL3MI_EVT37 This event measured by this field can be overwritten by AAC4[4].

PWRMTR_WT3_REVT41TO44

PWRMTR_WT3_REVT41TO44 - PWRMTR_WT3_REVT41TO44

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA28h

PWRMTR_WT3_REVT41to44

DWord	Bit	Description
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT3_RTAOI_EVT41** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTAOI_EVT41 This event measured by this field can be overwritten by AAC4[8].

PWRMTR_WT3_REVT49TO52

PWRMTR_WT3_REVT49TO52 - PWRMTR_WT3_REVT49TO52

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA30h

PWRMTR_WT3_REVT49to52

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_RTAOH_EVT52** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTAOH_EVT52 This event measured by this field can be overwritten by AAC4[19].
	23:16	**PWRMTR_WT3_RTAOH_EVT51** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTAOH_EVT51 This event measured by this field can be overwritten by AAC4[18].
	15:8	**Reserved** Default Value: 00h Access: RO Reserved
	7:0	**PWRMTR_WT3_RTAOI_EVT49** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTAOI_EVT49 This event measured by this field can be overwritten by AAC4[16].

PWRMTR_WT3_REVT53TO56

PWRMTR_WT3_REVT53TO56 - PWRMTR_WT3_REVT53TO56

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA34h

PWRMTR_WT3_REVT53to56

DWord	Bit	Description
0	31:24	**PWRMTR_WT3_RTSOI_EVT56** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTS OI_EVT56 This event measured by this field can be overwritten by AAC4[23].
	23:16	**PWRMTR_WT3_RTACN_EVT55** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTACN_EVT55 This event measured by this field can be overwritten by AAC4[22].
	15:8	**PWRMTR_WT3_RTACN_EVT54** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RTACN_EVT54 This event measured by this field can be overwritten by AAC4[21].
	7:0	**Reserved** Default Value: 00h Access: RO Reserved

PWRMTR_WT3_REVT65TO68

PWRMTR_WT3_REVT65TO68 - PWRMTR_WT3_REVT65TO68		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AA40h	
PWRMTR_WT3_REVT65to68		
DWord	**Bit**	**Description**
0	31:24	**Reserved** Default Value: 00h Access: RO Reserved
	23:16	**Reserved** Default Value: 00h Access: RO Reserved
	15:8	**PWRMTR_WT3_RGTCN_EVT66** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RGTCN_EVT66 This event measured by this field can be overwritten by AAC8[1].
	7:0	**PWRMTR_WT3_RGTOI_EVT65** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RGTOI_EVT65 This event measured by this field can be overwritten by AAC8[0].

PWRMTR_WT3_REVT69TO70

PWRMTR_WT3_REVT69TO70 - PWRMTR_WT3_REVT69TO70

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AA44h

PWRMTR_WT3_REVT69to70

DWord	Bit	Description
0	31:16	**Reserved** Default Value: 0000h Access: RO Reserved
	15:8	**PWRMTR_WT3_RGTCN_EVT70** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RGTCN_EVT70 This event measured by this field can be overwritten by AAC8[5].
	7:0	**PWRMTR_WT3_RGTCN_EVT69** Default Value: 00h Access: R/W Lock PWRMTR_WT3_RGTCN_EVT69 This event measured by this field can be overwritten by AAC8[4].

PWRMTREVTORE0

PWRMTREVTORE0 - PWRMTREVTORE0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AACCh
GT_CR_POWER_METER_EVENT_OVERRIDE_EU1_32	

DWord	Bit	Description
0	31:0	**PMEVORE0** Default Value: 00000000h Access: R/W Lock Power Meter Event Override for Render Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 1 through 32, where bit 0 corresponds to event 1, and bit 31 corresponds to event 32. "

PWRMTREVTORE1

PWRMTREVTORE1 - PWRMTREVTORE1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AAD0h

GT_CR_POWER_METER_EVENT_OVERRIDE_EU33_58

DWord	Bit	Description
0	31:26	**Reserved** Default Value: 00000000h Access: RO Reserved
	25:0	**PMEVORE1** Default Value: 00000000h Access: R/W Lock Power Meter Event Override for Render Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 33 through 64, where bit 0 corresponds to event 33, and bit 25 corresponds to event 58. "

PWRMTREVTORM0

PWRMTREVTORM0 - PWRMTREVTORM0		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AAD4h	
GT_CR_POWER_METER_EVENT_OVERRIDE_MEDIA1_32		
DWord	**Bit**	**Description**
0	31:0	**PMEVORM0** Default Value: 00000000h Access: R/W Lock Power Meter Event Override for Media Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 1 through 32, where bit 0 corresponds to event 1, and bit 31 corresponds to event 32. "

PWRMTREVTORM1

PWRMTREVTORM1 - PWRMTREVTORM1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0AAD8h

GT_CR_POWER_METER_EVENT_OVERRIDE_MEDIA33_36

DWord	Bit	Description
0	31:4	**Reserved** Default Value: 00000000h Access: RO Reserved
	3:0	**PMEVORM1** Default Value: 0h Access: R/W Lock Power Meter Event Override for Media Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 33 through 36, where bit 0 corresponds to event 33, and bit 3 corresponds to event 36. "

PWRMTREVTORR0

PWRMTREVTORR0 - PWRMTREVTORR0		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AAC0h	
GT_CR_POWER_METER_EVENT_OVERRIDE_RENDER1_32		
DWord	**Bit**	**Description**
0	31:0	**PMEVORR0** Default Value: 00000000h Access: R/W Lock Power Meter Event Override for Render Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 1 through 32, where bit 0 corresponds to event 1, and bit 31 corresponds to event 32.

PWRMTREVTORR1

PWRMTREVTORR1 - PWRMTREVTORR1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AAC4h	
GT_CR_POWER_METER_EVENT_OVERRIDE_RENDER33_64		
DWord	**Bit**	**Description**
0	31:0	**PMEVORR1** Default Value: 00000000h Access: R/W Lock Power Meter Event Override for Render Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 33 through 64, where bit 0 corresponds to event 33, and bit 31 corresponds to event 64.

PWRMTREVTORR2

PWRMTREVTORR2 - PWRMTREVTORR2		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0AAC8h	
GT_CR_POWER_METER_EVENT_OVERRIDE_RENDER65_70		
DWord	**Bit**	**Description**
0	31:6	**Reserved** Default Value: 00000000h Access: RO Reserved
	5:0	**PMEVORR2** Default Value: 00h Access: R/W Lock Power Meter Event Override for Render Events.Allows user to override signal coming in from Gfx engine and count event every cycle on a per event basis. For each event: 0: Power Meter works normally with event input taken from Gfx engine. 1: Force event input to weighting logic high every cuclk, overriding event data coming in from Gfx engine. This register is the override for events 65 through 70, where bit 0 corresponds to event 65, and bit 5 corresponds to event 70. "

RAM Clock Gating Control 1

RCGCTL1 - RAM Clock Gating Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09410h

RAM Clock Gating Control Registers.

DWord	Bit	Description
0	31	**USBunit RAM Clock Gating Disable** Access: R/W USBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**VLFunit RAM Clock Gating Disable** Access: R/W VLFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VISunit RAM Clock Gating Disable** Access: R/W VISunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**STCunit RAM Clock Gating Disable** Access: R/W STCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	27	**TDSunit RAM Clock Gating Disable** Access: R/W TDSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**VMCunit RAM Clock Gating Disable** Access: R/W VMCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**QRCunit RAM Clock Gating Disable** Access: R/W QRCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**SCunit RAM Clock Gating Disable** Access: R/W SCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**SVLunit RAM Clock Gating Disable** Access: R/W SVLunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**VFunit RAM Clock Gating Disable** Access: R/W VFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	21	**URBunit RAM Clock Gating Disable** Access: R/W URBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**GAMWunit RAM Clock Gating Disable** Access: R/W GAMWunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**SVGunit RAM Clock Gating Disable** Access: R/W SVGunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**RCZunit RAM Clock Gating Disable** Access: R/W RCZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**RCPBEunit RAM Clock Gating Disable** Access: R/W RCPBEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**RCCunit RAM Clock Gating Disable** Access: R/W RCCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	15	**PSDunit RAM Clock Gating Disable** Access: R/W PSDunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**MTunit RAM Clock Gating Disable** Access: R/W MTunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**SBEunit RAM Clock gating Disable** Access: R/W SBEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**IZunit RAM Clock Gating Disable** Access: R/W IZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**Reserved** Access: R/W
	10	**ICunit RAM Clock Gating Disable** Access: R/W ICunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**HIZunit RAM Clock Gating Disable** Access: R/W HIZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

8	**GAMunit RAM Clock Gating Disable** Default Value: 0b Access: R/W GAMunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
7	**BCunit RAM Clock Gating Disable** Access: R/W BCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
6	**HDCunit RAM Clock Gating Disable** Access: R/W GAFSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
5	**DMunit RAM Clock Gating Disable** Access: R/W DMunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
4	**WMFEunit RAM Clock Gating Disable** Access: R/W WMFEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
3	**CSunit RAM Clock Gating Disable** Access: R/W CSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	2	**BLBunit RAM Clock Gating Disable** Access: R/W BLBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**MPCunit RAM Clock Gating Disable** Access: R/W MPCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**BFunit RAM Clock Gating Disable** Access: R/W BFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RAM Clock Gating Control 1

RCGCTL1 - RAM Clock Gating Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000100
Size (in bits):	32
Address:	09410h

RAM Clock Gating Control Registers.

DWord	Bit	Description
0	31	**USBunit RAM Clock Gating Disable** Access: R/W USBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**VLFunit RAM Clock Gating Disable** Access: R/W VLFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VISunit RAM Clock Gating Disable** Access: R/W VISunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**STCunit RAM Clock Gating Disable** Access: R/W STCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	27	**TDSunit RAM Clock Gating Disable** Access: R/W TDSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**VMCunit RAM Clock Gating Disable** Access: R/W VMCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**QRCunit RAM Clock Gating Disable** Access: R/W QRCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**SCunit RAM Clock Gating Disable** Access: R/W SCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**SVLunit RAM Clock Gating Disable** Access: R/W SVLunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**VFunit RAM Clock Gating Disable** Access: R/W VFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	21	**URBunit RAM Clock Gating Disable** Access: R/W URBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**GAMWunit RAM Clock Gating Disable** Access: R/W GAMWunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**SVGunit RAM Clock Gating Disable** Access: R/W SVGunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**RCZunit RAM Clock Gating Disable** Access: R/W RCZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**RCPBEunit RAM Clock Gating Disable** Access: R/W RCPBEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**RCCunit RAM Clock Gating Disable** Access: R/W RCCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	15	**PSDunit RAM Clock Gating Disable** Access: R/W PSDunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**MTunit RAM Clock Gating Disable** Access: R/W MTunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**SBEunit RAM Clock gating Disable** Access: R/W SBEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**IZunit RAM Clock Gating Disable** Access: R/W IZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**Reserved** Access: R/W
	10	**ICunit RAM Clock Gating Disable** Access: R/W ICunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**HIZunit RAM Clock Gating Disable** Access: R/W HIZunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	8	**GAMunit RAM Clock Gating Disable** Default Value: 1b Access: R/W GAMunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**BCunit RAM Clock Gating Disable** Access: R/W BCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**HDCunit RAM Clock Gating Disable** Access: R/W GAFSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**DMunit RAM Clock Gating Disable** Access: R/W DMunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**WMFEunit RAM Clock Gating Disable** Access: R/W WMFEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**CSunit RAM Clock Gating Disable** Access: R/W CSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL1 - RAM Clock Gating Control 1

	2	**BLBunit RAM Clock Gating Disable** Access: R/W BLBunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**MPCunit RAM Clock Gating Disable** Access: R/W MPCunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**BFunit RAM Clock Gating Disable** Access: R/W BFunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RAM Clock Gating Control 2

RCGCTL2 - RAM Clock Gating Control 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x7FC00000
Size (in bits):	32
Address:	09414h

RAM Clock Gating Control Registers.

DWord	Bit	Description
0	31	**SPARE 2 clock gate disable** Access: R/W SPARE 2 unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30:28	**VMCRunit clock gate disable** Default Value: 111b Access: R/W VMCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27:25	**SMCRunit clock gate disable** Default Value: 111b Access: R/W SMCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24:22	**MCRunit clock gate disable** Default Value: 111b Access: R/W MCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	21	**Reserved**
	20	**WVISunit clock gate disable** Access: R/W WVIS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**WAVM unit RAM clock gate disable** Access: R/W WAVM unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**WHME unit RAM clock gate disable bit** Access: R/W WHME unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**WIME unit RAM clock gate disable** Access: R/W WIME unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**WMPC unit RAM clock gating disable** Access: R/W WMPC unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**SDEunit RAM clock gate disable** Access: R/W SDE unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	14	**VSHM unit clock gate disable** Access: R/W VSHM unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**DAPRTS unit RAM clock gate disable** Access: R/W DAPRTS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled.
	12	**GS unit RAM clock gate disable** Access: R/W GS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**Reserved**
	10	**GAMTunit RAM clock gate disable bit** Access: R/W GAMT unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VCW unit RAM clock gate disable** Access: R/W VCW unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**VEO unit RAm clock gate disable** Access: R/W VEO unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	7	**IMEunit RAM clock gate disable** Access: R/W IMEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**CREunit RAM clock gate disable** Access: R/W CREunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**RSunit RAM clock gate disable** Access: R/W RSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**MSCunit RAM Clock Gating Disable** Access: R/W MSCunit RAM Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**VMXunit RAM Clock Gating Disable** Access: R/W VMXunit RAM Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**GAunit RAM Clock Gating Disable for all EUs** Access: R/W GAunit RAM Clock Gating Disable Control For all EUs in each Row: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	1	**VSunit RAM Clock Gating Disable** Access: R/W VSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**HSunit RAM Clock Gating Disable** Access: R/W HSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RAM Clock Gating Control 2

RCGCTL2 - RAM Clock Gating Control 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0xFFC00000
Size (in bits):	32
Address:	09414h

RAM Clock Gating Control Registers.

DWord	Bit	Description
0	31	**1x2X Assign fub XOR clock gate disable** Default Value: 1b Access: R/W XOR based unit level clock gating disable in 1x2x_asgn fub: '0' : XOR Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : XOR Clock Gating Disabled. (i.e., clocks are toggling, always)
	30:28	**VMCRunit clock gate disable** Default Value: 111b Access: R/W VMCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27:25	**SMCRunit clock gate disable** Default Value: 111b Access: R/W SMCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24:22	**MCRunit clock gate disable** Default Value: 111b Access: R/W MCR unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	21	**MUCunit RAM clock gate disable** Access: R/W MUC unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**WVISunit clock gate disable** Access: R/W WVIS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**WAVM unit RAM clock gate disable** Access: R/W WAVM unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**WHME unit RAM clock gate disable bit** Access: R/W WHME unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**WIME unit RAM clock gate disable** Access: R/W WIME unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**WMPC unit RAM clock gating disable** Access: R/W WMPC unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	15	**SDEunit RAM clock gate disable** Access: R/W SDE unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**VSHM unit clock gate disable** Access: R/W VSHM unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**DAPRTS unit RAM clock gate disable** Access: R/W DAPRTS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled.
	12	**GS unit RAM clock gate disable** Access: R/W GS unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**Reserved** Access: R/W
	10	**GAMTunit RAM clock gate disable bit** Access: R/W GAMT unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VCW unit RAM clock gate disable** Access: R/W VCW unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	8	**VEO unit RAm clock gate disable** Access: R/W VEO unit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**IMEunit RAM clock gate disable** Access: R/W IMEunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**CREunit RAM clock gate disable** Access: R/W CREunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**RSunit RAM clock gate disable** Access: R/W RSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**MSCunit RAM Clock Gating Disable** Access: R/W MSCunit RAM Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**VMXunit RAM Clock Gating Disable** Access: R/W VMXunit RAM Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RCGCTL2 - RAM Clock Gating Control 2

	2	**GAunit RAM Clock Gating Disable for all EUs** Access: R/W GAunit RAM Clock Gating Disable Control For all EUs in each Row: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**VSunit RAM Clock Gating Disable** Access: R/W VSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**HSunit RAM Clock Gating Disable** Access: R/W HSunit RAM Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RAM Clock Gating Control 3

RCGCTL3 - RAM Clock Gating Control 3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09434h

DWord	Bit	Description
0	31:13	**RSVD** Access: RO Reserved
	12	**Reserved**
	11	**cp_ramcgdis_hwm** Access: R/W HWM unit Clock Gating Disable (cp_ramcgdis_hwm) HWM unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**cp_ramcgdis_hed** Access: R/W HED unit Clock Gating Disable (cp_ramcgdis_hed) HED unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**cp_ramcgdis_hpp** Access: R/W HPP unit Clock Gating Disable (cp_ramcgdis_hpp) HPP unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**cp_ramcgdis_hpr** Access: R/W HPR unit Clock Gating Disable (cp_ramcgdis_hpr) HPR unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality)

RCGCTL3 - RAM Clock Gating Control 3

		'1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**cp_ramcgdis_hmc** Access: R/W HMC unit Clock Gating Disable (cp_ramcgdis_hmc) HMC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**cp_ramcgdis_hlf** Access: R/W HLF unit Clock Gating Disable (cp_ramcgdis_hlf) HLF unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**cp_ramcgdis_hmx** Access: R/W HMX unit Clock Gating Disable (cp_ramcgdis_hmx) HMX unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**cp_ramcgdis_vmm** Access: R/W VMM unit Clock Gating Disable (cp_ramcgdis_vmm) VMM unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**cp_ramcgdis_mpd** Access: R/W MPD unit Clock Gating Disable (cp_ramcgdis_mpd) MPD unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality)

RCGCTL3 - RAM Clock Gating Control 3

		'1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**cp_ramcgdis_mbd** Access: R/W MBD unit Clock Gating Disable (cp_ramcgdis_mbd) MBD unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**cp_ramcgdis_mmx** Access: R/W MMX unit Clock Gating Disable (cp_ramcgdis_mmx) MMX unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**cp_ramcgdis_vmpc** Access: R/W VMPC unit Clock Gating Disable (cp_ramcgdis_vmpc) VMPC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

RC6 Wake Rate Limit

RCXWRL - RC6 Wake Rate Limit		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A09Ch-0A09Fh	

DWord	Bit	Description
0	31:16	**RC6WRL** Access: R/W RC Promotion Time Modulated by wake rate limits when using EI method: Even though the RC6 promotion time is met, if the wake limit is exceeded, no promotion pmcr_rc6_wake_rate_limit[15:0]
	15:0	**Reserved** Access: RO

RCC LRA 0

RCC_LRA_0 - RCC LRA 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x7F403F00
Size (in bits):	32
Address:	04A40h

DWord	Bit	Description
0	31:24	**RCC LRA1 Max** Default Value: 01111111b Access: R/W Maximum value of programmable LRA1.
	23:16	**RCC LRA1 Min** Default Value: 01000000b Access: R/W Minimum value of programmable LRA1.
	15:8	**RCC LRA0 Max** Default Value: 00111111b Access: R/W Maximum value of programmable LRA0.
	7:0	**RCC LRA0 Min** Default Value: 00000000b Access: R/W Minimum value of programmable LRA0.

RCC LRA 1

RCC_LRA_1 - RCC LRA 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001
Size (in bits):	32
Address:	04A44h

DWord	Bit	Description
0	31:2	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	1	**MSC LRA** Default Value: 0b Access: R/W Which LRA should MSC use.
	0	**RCC LRA** Default Value: 1b Access: R/W Which LRA should RCC use.

RCC Virtual page Address Registers

RCCTLB_VA - RCC Virtual page Address Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04A00h-04A03h

These registers are directly mapped to the current Virtual Addresses in the RCCTLB (Render Cache for Color TLB).

DWord	Bit	Description
0	31:12	**Address** Project: All Format: GraphicsAddress[31:12] Page virtual address.
	11:0	**Reserved** Project: CHV, BSW Format: MBZ

RC Evaluation Interval

RCI - RC Evaluation Interval		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A0A8h-0A0ABh	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO
	23:0	**Render Standby Evaluation Interval** Access: R/W 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_rc_ei[23:0]

RC Idle Hysteresis

RCIHYST - RC Idle Hysteresis		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A0ACh-0A0AFh	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO
	23:0	**RC Idle Hysteresis Detection** Access: R/W Idle intervals must be longer than this value to be considered idle. 0 = 0 usec means disabled 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec This must not be set to more than 5ms to prevent the PCU from timing out on an S state entry

RCS_PREEMPTION_HINT

RCS_PREEMPTION_HINT - RCS_PREEMPTION_HINT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	024BCh

Description

This register contains the Head pointer offset in to the Ring Buffer or the Dword aligned Graphics address in to the Batch Buffer corresponding to either MI_ARB_CHECK called Preemption Hint Address. When Preemption Hint Address is enabled, RCS will honor UHPTR only on parsing MI_ARB_CHK at Preemption Hint Address.

This register contains the Head pointer offset into the Ring Buffer or the Dword aligned Graphics address into the Batch Buffer corresponding to the below listed commands in execlist mode of operation

- MI_ARB_CHECK
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT
- 3D_PRIMITIVE
- GPGPU_WALKER
- MEDIA_STATE_FLUSH
- PIPE_CONTROL (Only in GPGPU mode of pipeline selection)
- MI_ATOMIC (Post Sync Operation set in GPGPU mode of pipeline selection)
- MI_SEMAPHORE_SIGNAL (Post Sync Operation set in GPGPU mode of pipeline selection)

Programming Notes

Programming Restriction:
This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHECK in command stream. Programmer has to ensure that RCS Preemption Hint register gets programmed before UHPTR is programmed and well before RCS crosses the corresponding execution point. Preemption hint for both RingBuffer and Batch Buffer can't be enabled simultaneously.

User must ensure the Preempted Hint Address programmed matches either Ring Head Offset or Batch Buffer Graphics Virtual Address and not both of them.
User must also ensure the Preempted Hint Address[19:0] programmed matches either Ring Head Offset[19:0] or Batch Buffer Graphics Virtual Address[19:0] and not both of them.

RCS_PREEMPTION_HINT - RCS_PREEMPTION_HINT

DWord	Bit	Description
0	31:2	**Preempted Hint Address** Project: CHV, BSW Format: U30 This field contains the Head offset in to the Ring Buffer when Preemption Hint is set to Ring Buffer and Dword aligned Graphics Address in to the batch buffer when Preemption Hint is set to Batch Buffer.
	1	**Batch Buffer Preemption Hint** Project: CHV, BSW Format: Enabled Value / Name / Description / Project: 0h / Disabled / Preemption hint is disabled in batch buffer. / CHV, BSW 1h / Enabled / Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Batch Buffer. / CHV, BSW
	0	**Ring Preemption Hint** Project: CHV, BSW Format: Enable Value / Name / Description / Project: 0h / Disable / Preemption hint is disabled in ring buffer. / CHV, BSW 1h / Enabled / Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Ring Buffer. / CHV, BSW

RCS_PREEMPTION_HINT_UDW

RCS_PREEMPTION_HINT_UDW - RCS_PREEMPTION_HINT_UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	024C8h

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the Batch Buffer corresponding to MI_ARB_CHECK command called Preemption Hint Address.

Programming Notes

Programming Restriction:
This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHK in command stream.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Preempted Hint Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer when Preemption Hint is set to Batch Buffer. This field is not valid when Preemption Hint is set to Ring Buffer.

RCS Batch Buffer State Register

RCS_BB_STATE - RCS Batch Buffer State Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000 CHV, BSW
Access:	RO
Size (in bits):	32
Address:	02110h

This register contains the attributes of the current batch buffer initiated from the Ring Buffer.

This register should not be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer. This register is saved and restored with context.

DWord	Bit	Description
0	31:9	**Reserved** Format: MBZ
	8	**Reserved** Format: MBZ
	7	**Resource Streamer Enable** Project: CHV, BSW Format: U1 When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer.
	6	**Reserved**
	5	**Address Space Indicator** Project: CHV, BSW Note: This field reflects the effective address space indicator security level and may not be the same as the Address Space Indicator written using MI_BATCH_BUFFER_START. Value / Name / Description 0h / GGTT **[Default]** / This Batch buffer is located in GGTT memory and is privileged 1h / PPGTT / This Batch buffer is located in PPGTT memory and is non-privileged.
	4	**Reserved** Project: CHV, BSW Format: MBZ
	3:0	**Reserved** Format: MBZ

RCS Context Preemption Hint

RCS_CTXID_PREEMPTION_HINT - RCS Context Preemption Hint

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	024CCh
Address:	124CCh-124CFh
Name:	Context ID Preemption Hint
ShortName:	RCS_CTXID_PREEMPTION_HINT_VCSUNIT0
Address:	1A4CCh-1A4CFh
Name:	Context ID Preemption Hint
ShortName:	RCS_CTXID_PREEMPTION_HINT_VECSUNIT
Address:	1C4CCh-1C4CFh
Name:	Context ID Preemption Hint
ShortName:	RCS_CTXID_PREEMPTION_HINT_VCSUNIT1
Address:	224CCh-224CFh
Name:	Context ID Preemption Hint
ShortName:	RCS_CTXID_PREEMPTION_HINT_BCSUNIT

This register contains the Context ID of a context in execlist mode of operation. In execlist mode of operation RCS_PREEMPTION_HINT and RS_PREEMPTION_HINT registers are looked at by Render Command Streamer and Resource Streamer only on executing a context having Context ID that matches with the contents of this register. This register contents are valid and looked at only in execlist mode of operation

Programming Restriction:

This register should NEVER be programmed in functional mode, this must be used only in validation mode to achieve deterministic preemption behavior in execlist mode of operation.

DWord	Bit	Description
0	31:0	**Context ID Preemption Hint** Format: U32 If 0 this field has no effect. If nonzero it indicates the only context ID that can be preempted when execlists are enabled. A preemption attempt when the context ID of the currently executing ring context does not match this field will be ignored.

RC Wake Counter

RCWC - RC Wake Counter

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A0A4h-0A0A7h

DWord	Bit	Description
0	31:16	**Wake Counter Render** Access: RO Incremented for each wake event , wraps around wake_counter[15:0]
	15:0	**Wake Counter Media** Access: RO Incremented for each wake event , wraps around wake_counter[15:0]

RCZ Virtual Page Address Registers

RCZTLB_VA - RCZ Virtual Page Address Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04B00h-04B03h

These registers are directly mapped to the current Virtual Addresses in the RCZTLB (Render Cache for Z (Depth), Hi Z, and Stencil TLB).

DWord	Bit	Description
0	31:12	**Address** Format: GraphicsAddress[31:12] Page virtual address.
	11:0	**Reserved** Format: MBZ

Ready Bit Vector 0 for TLBPEND registers

TLBPEND_RDY0 - Ready Bit Vector 0 for TLBPEND registers	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04708h-0470Bh

This register contains the ready bits for entries 0-31 of TLBPEND structure (Cycles pending TLB translation).

DWord	Bit	Description
0	31:0	**Ready bits per entry**

Ready Bit Vector 1 for TLBPEND registers

TLBPEND_RDY1 - Ready Bit Vector 1 for TLBPEND registers	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	0470Ch-0470Fh

This register contains the ready bits for entries 32-63 of TLBPEND structure (Cycles pending TLB translation).

DWord	Bit	Description
0	31:0	**Ready bits per entry**

Render C State Control 1

RCCTL1 - Render C State Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A090h-0A093h

DWord	Bit	Description
0	31:29	**Reserved** Access: RO
	28	**TO - HW RC Promotion (i.e., Depth) Selection** Access: R/W 0: Timeout method disabled 1: TO method enabled pmcr_to_enable
	27	**EI - HW RC Promotion (i.e., Depth) Selection** Access: R/W 0: EI disabled 1: Evaluation Interval (EI) method enabled pmcr_ei_enable
	26:25	**Reserved** Access: RO
	24	**Reserved** Access: R/W
	23	**Reserved** Access: R/W
	22:0	**Reserved** Access: RO

Render forcewake acknowledge

RENFW_ACK - Render forcewake acknowledge

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300B4h

This register contains the per thread force wake acknowledge bits for the Render power well.

DWord	Bit	Description
0	31:16	**RESERVED** Default Value: 0000h Access: RO Reserved.
	15:0	**FWAKERENDERACK** Default Value: 0000h Access: RO Force Wake Render request bits. Driver must poll on the corresponding bit to confirm that the well has woken. For example, if 13_00B0[0] is written to a '1' (along with 13_00B0[16]='1'), then bit0 of this register indicates when the force wake request has been completed.

Render forcewake request

RENFW_REQ - Render forcewake request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	1300B0h

This register contains per thread force wake request bits for the Render power Well. The upper 16 bits act as masks for the lower 16 bits. Bit 31 masks bit 15 and bit 16 masks bit 0.

1. Driver writes to GPM force wake request bit. (VV will have a Render(13_00B0(15:0)) and a Media (13_00B8(15:0)) bits.)
2. The GPM responds by writing (via PLINK) to 1300B4[15:0] or 1300BC[15:0] register.
3. Driver polls (1300B4[15:0] and/or 1300BC[15:0])status until 1... indicating that that well has completed wake sequence.

Since the registers are per thread, only the specific bit that was forced should be checked for status.

DWord	Bit	Description
0	31:16	**FWAKERENDERREQMSK** Default Value: 0000h Access: RO Mask bits for lower 16 bits to avoid a read modify/write. If '0', the corresponding bit in [15:0] is not changed. If '1', the corresponding bit in [15:0] is changed to the value in [15:0]
	15	**FWAKERENDERREQ15** Default Value: 0b Access: R/W Force Wake Render request 15.
	14	**FWAKERENDERREQ14** Default Value: 0b Access: R/W Force Wake Render request 14.
	13	**FWAKERENDERREQ13** Default Value: 0b Access: R/W Force Wake Render request 13.

RENFW_REQ - Render forcewake request

	12	**FWAKERENDERREQ12** Default Value: 0b Access: R/W Force Wake Render request 12.
	11	**FWAKERENDERREQ11** Default Value: 0b Access: R/W Force Wake Render request 11.
	10	**FWAKERENDERREQ10** Default Value: 0b Access: R/W Force Wake Render request 10.
	9	**FWAKERENDERREQ9** Default Value: 0b Access: R/W Force Wake Render request 9.
	8	**FWAKERENDERREQ8** Default Value: 0b Access: R/W Force Wake Render request 8.
	7	**FWAKERENDERREQ7** Default Value: 0b Access: R/W Force Wake Render request 7.
	6	**FWAKERENDERREQ6** Default Value: 0b Access: R/W Force Wake Render request 6.
	5	**FWAKERENDERREQ5** Default Value: 0b Access: R/W Force Wake Render request 5.

RENFW_REQ - Render forcewake request

	4	**FWAKERENDERREQ4** Default Value: 0b Access: R/W Force Wake Render request 4.
	3	**FWAKERENDERREQ3** Default Value: 0b Access: R/W Force Wake Render request 3.
	2	**FWAKERENDERREQ2** Default Value: 0b Access: R/W Force Wake Render request 2.
	1	**FWAKERENDERREQ1** Default Value: 0b Access: R/W Force Wake Render request 1.
	0	**FWAKERENDERREQ0** Default Value: 0b Access: R/W Force Wake Render request 0.

Render Geyserville Mode Control 1

RPMODECTL1 - Render Geyserville Mod

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A024h-0A027h

DWord	Bit	Description
0	31:12	**Reserved** Access:
	11	**RP Video Turbo Enable for Media Engine** Access: Refer to bits A024[10:9] for different Media turbo scenarios. Refer to the table in the description for bits A024[10:9].
	10:9	**RP Software Mode Control** Access: 11 - Media_active (8000[6]), Media_done and RPVTENMEDIA(A024[11]) are don't care. Media turbo freq is sele 10 - Media_active (8000[6]), Media_done and RPVTENMEDIA(A024[11]) are don't care. Normal freq(Hard codec 01 - Based on the status either the media-turbo or normal frequency will be selected. If Media_active(8000[6]) is set, Media_done and RPVTENMEDIA(A024[11]) are don't care. Media turbo freq is se Else If RPVTENMEDIA(A024[11]) is 0, Media_done is don't care. Normal freq is selected. Else If RPVTENMEDIA(A024[11]) is 1, Media turbo freq is selected based on internal status. if Media_done is set 00 - Media_active (8000[6]), Media_done and RPVTENMEDIA(A024[11]) are don't care. Current frequency is ma Note: Media is done only when both VCS(8004[0] =1) and VECS(8010[0] = 1) are idle. Media_done = 8004[0] a

pmcr_rpsw_ctl_mode[1]	pmcr_rpsw_ctl_mode[0]	pmmr_media_active	pmcr_rpsw_vten_vcs	pmmr_vcs_
A024[10]	A024[9]	8000[4]	A024[11]	8000[1]
1	1	x	x	x
1	0	x	x	x
0	1	1	x	x
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0

RPMODECTL1 - Render Geyserville Mod

0	1	0	1	1
0	0	x	x	x

CHV, BSW description

pmcr_rpsw_ctl_mode[1]	pmcr_rpsw_ctl_mode[0]	pmmr_media_active	pmcr_rpsw_vten_vcs	pmmr_vcs
A024[10]	A024[9]	8000[6]	A024[11]	8004[0]
1	1	x	x	x
1	0	x	x	x
0	1	1	x	x
0	1	0	0	0
0	1	0	0	1
0	1	0	1	0
0	1	0	1	1
0	0	x	x	x

8 **Mask Bits for Graphics Busyness**

Access:

Description
Mask Bits for Graphics Busyness (MBGB) and enables MC0 counter: 0: MFX busyness is not counted as part of gfx busyness and media MC0 counter forced to a zero. 1: MFX busyness is counted as part of gfx busyness and media MC0 counter is allowed to count. Both VCS and VECS is counted towards media busyness. BIOS/driver should always set this bit to a one. pmcr_media_mask
Mask Bits for Graphics Busyness (MBGB): 0: MFX busyness is not counted as part of gfx busyness 1: MFX busyness is counted as part of gfx busyness Both VCS and VECS is counted towards media busyness pmcr_media_mask

7 **Render Geyserville HW Controlled Idle Mode Enable**

Access:

0 - Disables Render Geyserville (RP) function (default).

1 - Turns on the Render Geyserville (RP) function and RP counters are enabled (program enable after counter v

RPMODECTL1 - Render Geyserville Mod

6	**Reserved** Access: Reserved
5:3	**Frequency Increase Utilization Metric Selection** Access: Frequency Increase Utilization Metric selection The selection of a metric below indicates which one is to be used in making decision. All the metrics should be More than Busy Max Continuous (BMXC) time must be reached for a frequency increase More than Busy Max Average (BMXA) at end of Evaluation Interval must be reached for a frequency increase Less than Idle Min Continuous (IMNC) time must be reached for a frequency Increase BMXC BMXA IMNC 0 0 0 : no metric enabled 0 0 1 : IMNC metric enabled (0 % val) 0 1 0 : BMXA metric enabled (90 % val) 0 1 1 : reserved 1 0 0 : BMXC metric enabled (10 % val) 1 0 1 : reserved 1 1 0 : reserved 1 1 1 : reserved pmcr_freq_inc_utimet[2:0]
2:0	**Frequency Decrease Utilization Metric Selection** Access: The selection of a metric below indicates which one is to be used in making decision. All the metrics should be Less than Busy Min Continuous must be reached for a frequency decrease Less than Busy Min Average at end of Evaluation Interval must be reached for a frequency decrease More than Idle Max Continuous must be reached for a frequency decrease BMNC BMNA IMXC 0 0 0 : no metric enabled 0 0 1 : IMXC metric enabled (0 % val) 0 1 0 : BMNA metric enabled (100 % val) 0 1 1 : reserved 1 0 0 : BMNC metric enabled (0% val) 1 0 1 : reserved 1 1 0 : reserved 1 1 1 : reserved pmcr_freq_dec_utimet[2:0]

Render Mode Register for Software Interface

MI_MODE - Render Mode Register for Software Interface

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0209Ch
Address:	1209Ch-1209Fh
Name:	Mode Register for Software Interface
ShortName:	MI_MODE_VCSUNIT0
Address:	1A09Ch-1A09Fh
Name:	Mode Register for Software Interface
ShortName:	MI_MODE_VECSUNIT
Address:	1C09Ch-1C09Fh
Name:	Mode Register for Software Interface
ShortName:	MI_MODE_VCSUNIT1
Address:	2209Ch-2209Fh
Name:	Mode Register for Software Interface
ShortName:	MI_MODE_BCSUNIT

The MI_MODE register contains information that controls software interface aspects of the Memory Interface function.

DWord	Bit	Description
0	31:16	**Masks** Format: Mask[15:0] A 1 in a bit in this field allows the modification of the corresponding bit in Bits 15:0
	15	**Suspend Flush** Format: U1

Value	Name	Description
0h	No Delay **[Default]**	HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well
1h	Delay Flush	Suspend flush is active

MI_MODE - Render Mode Register for Software Interface

Programming Notes
This should only be written to from the ring using MI_SUSPEND_FLUSH. It is considered undefined if written by software through MMIO

14 **Async Flip Performance mode**

Format:	U1

Value	Name	Description
0h	Performance mode enabled **[Default]**	The stall of the flip event is in the windower
1h	Performance mode disabled	The stall of the flip event is in the command stream

Programming Notes
This bit must be set to '1' on all projects disabling Async Flip Performance mode.
When Async Flip Performance mode is enabled stall is in the Windower allowing the commands following the MI_WAIT_FOR_EVENT to be parsed by command streamer, this breaks the usage model of controlling the display message generation in display engine using MI_LOAD_REGISTER_IMMEDIATE commands from ring buffer.

13 **Flush Performance mode**

Format:	U1

Value	Name	Description
0h	run fast restore **[Default]**	No NonPipelined SV flush.
1h	run slow legacy restore	With NonPipelined SV flush.

12 **Reserved**

Format:	MBZ

11 **Invalidate UHPTR enable**

Format:	Enable

If bit set H/W clears the valid bit of UHPTR (2134h, bit 0) when current active head pointer is equal to UHPTR.

10 **Atomic Read Return for MI_COPY_MEM_MEM**

Project:	CHV, BSW
Format:	U1

Value	Name	Description
0h	Disable **[Default]**	Hardware does a regular memory fence write to complete the write to the destination address before moving to the next instruction.
1h	Enable	Hardware does Atomic Move with Read Return to complete the write to the destination address before moving to the next instruction.

MI_MODE - Render Mode Register for Software Interface

9 **Rings Idle**

Format:	U1

Read Only Status bit

Value	Name	Description
0h	Not Idle **[Default]**	Parser not Idle or Ring Arbiter not Idle.
1h	Idle	Parser Idle and Ring Arbiter Idle.

Programming Notes
Writes to this bit are not allowed.

8 **Stop Rings**

Format:	U1

Value	Name	Description
0h	**[Default]**	Normal Operation.
1h		Parser is turned off and Ring arbitration is turned off.

Programming Notes
Software must set this bit to force the Rings and Command Parser to Idle. Software must read a 1 in the Ring Idle bit after setting this bit to ensure that the hardware is idle.
Software must clear this bit for Rings to resume normal operation.
Due to known HW issue when Stop Rings occur during execution of a batch buffer, memory access type of the batch buffer is reset and hence on resuming the memory access type can be inconsistent with the desired memory access type. SW must not set/reset Stop Rings to achieve stall and resume function in command streamer execution, however Stop Rings can be used by SW before resetting the engine.

7 **Reserved**

Format:	MBZ

6 **Vertex Shader Timer Dispatch Enable**

Project:	CHV, BSW
Format:	Enable

Value	Name	Description
0h	Disable **[Default]**	Disable the timer for dispatch of single vertices from the vertex shader. Vertex shader will try to collect 2 vertices before a dispatch
1h	Enable	Enable the timer for dispatch of single vertices. Dispatch a single vertex shader thread after the timer expires.

5 **Reserved**

Format:	MBZ

MI_MODE - Render Mode Register for Software Interface

4:1 **Predicate Enable**

Project:	CHV, BSW

This field gets set when "MI_SET_PREDICATE" command is parsed by render command streamer. Predicate Disable is the default mode of operation.

Value	Name	Description
0h	Predicate Disable	Predication is Disabled and RCS will process commands as usual.
1h	Predicate on Result2 clear	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is clear.
2h	Predicate on Result2 set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT_2 is set.
3h	Predicate on Result clear	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is clear.
4h	Predicate on Result set	Following Commands will be NOOPED by RCS only if the MI_PREDICATE_RESULT is set.
5h	Predicate when two or more slices enabled	Following Commands will be NOOPED by RCS only when one slice is enabled, NOOPED when more than one slice is enabled.
6h	Predicate when one or three slices enabled	Following Commands will be Executed by RCS only when two slices are enabled, NOOPED when one or three slices are enabled.
7h	Predicate when one or two slices enabled	Following Commands will be Executed by RCS only when all the three slices are enabled, NOOPED when less than three slices are enabled.
8h-Eh	Reserved	
Fh	Predicate Always	Following Commands will be NOOPED by RCS unconditionally.

Programming Notes
SW must use MI_SET_PREDICATE instead of MMIO access.

0 **Mask IIR disable**

Format:	Disable

Mask IIR disable. Nominally the Interrupt controller masks interrupts in the IIR register if an interrupt acknowledge from the 3gio interface is pending. Setting this bit to a 1 allows interrupts to be visible to the interrupt controller while an interrupt acknowledge is pending.

Render Performance Status 1

RPSTAT1 - Render Performance Status 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A01Ch-0A01Fh

This register reflects real-time values and thus will not have a pre-determined default value out of reset.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15	**Last Requested Video Turbo Mode** Access: RO Last Requested Video Turbo Mode (CRTM): 0 = Most recent request was a normal request (from RPNSWREQ) 1 = Most recent request was a Video Turbo request (from RPVSWREQ)
	14:8	**Reserved** Access: RO
	7:0	**Reserved** Access: RO

Render Performance Status Register

RP_STATUS0 - Render Performance Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A01Ch

This register reflects real-time values and thus does not have a pre-determined default value out of reset.

DWord	Bit	Description
0	31:20	**Reserved** Access: RO
	19:18	**Current Actual Gear Ratio** Access: RO
	17:16	**Previous Actual Gear Ratio** Access: RO Previous Actual Gear Ratio (PAGR).
	15	**Last Requested Video Turbo Mode** Access: RO Last Requested Video Turbo Mode (CRTM): 0 = Most recent request was a normal request (from RPNSWREQ). 1 = Most recent request was a Video Turbo request (from RPVSWREQ).
	14	**Reserved** Access: RO
	13:7	**Current Actual GFX Freq** Access: RO This is the MLC ratio that the core is actually running.
	6:0	**Previous Actual GFX Freq** Access: RO This is the MLC ratio that the core was actually running before the current actual GFX frequency.

Render Power Clock State Register

R_PWR_CLK_STATE - Render Power Clock State Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000288 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	020C8h

This register contains the mode selection for configuring render engine to attain desired performance and power requirements for a given context. This register is render context save/restored. This register must be initialized correctly when the context is submitted for the first time. This register is context save/restored as part of Exec-List context image in both Exec-List and Ring-Buffer mode of scheduling. This register contents are valid only when "Enable" bit [31] of the register is set.

Programming Notes

This register must not be programmed directly through CPU MMIO cycle. Exec-List Scheduling Mode: Every context can have its own required render engine configuration by programming this register appropriately in the logical render context image in memory (LRCA) before submitting the context to the execlist submit port. This register must not be programmed using MI_LOAD_REGISTER_IMM command in ring buffer or in batch buffer, however programming "NON-SLM Indication" field through MI_LOAD_REGISTER_IMM is an exception defined below. If a need arises to change the render configuration for a context being executed in HW, Scheduler must preempt the context and update the desired render configuration in the logical render context image in memory and resubmit the context. Only "NON-SLM Indictaion" field in R_PWR_CLK_STATE register is allowed to be modified through MI_LOAD_REGISTER_IMM command in ring_buffer or privileged_batch_buffer. SW must modify only "NON-SLM Indication" field and must ensure to program other fields with the same value as in LRCA. SW must ensure to program PIPECONTROl flush command with CS Stall and HDC Flush prior to programming MI_LOAD_REGISTER_IMM command to modify "NON-SLM Indication" in R_PWR_CLK_STATE register. Example: //R_PWR_CLK_STATE register value in LRCA configured with two slices and NON-SLM indication reset: 0x80005_0000 //SW desires to set NON-SLM Indication filed in ring buffer MI_LOAD_REGISTER_IMM 0x20C8, 0x8005_0100 Ring Buffer Scheduling: This register must be programmed using MI_LOAD_REGISTER_IMM command in the ring buffer. When this register is being programmed to re-configure the number of slices, SW must context save the state before programming this register and restore the state after programming the register via dummy MI_SET_CONTEXT command, this will ensure the existing state is programmed to all the new slices that are powered up, in case of slice shutdown this is not required. EX: MI_SET_CONTEXT → CXTA MI_BATCH_BUFFER_START MI_BATCH_BUFFER_START MI_SET_CONTEXT → CXTB //Dummy Context to save existing render state to be restored latter. MI_LOAD_REGISTER_IMM : R_PWR_CLK_STATE (1 Slice to 3 Slices) // Slice configuration done. MI_SET_CONTEXT → CXTA // Context restore of valid state to all the slices powered up.

R_PWR_CLK_STATE - Render Power Clock State Register

DWord	Bit	Description
0	31	**Power Clock State Enable** Project: CHV, BSW Format: U1 Value / Name / Description: 0h – Power Clock State Disabled – No specific power state set, bits[30:0] are ignored. 1h – Power Clock State Enabled – Power Clock is set and bit[30:0] are valid and have the desired state.
	30:0	**Render Power Clock State** Project: CHV, BSW Format: Power Clock State Format CHV, BSW

Render Power Meter Counter

RPMCNT - Render Power Meter Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A274h-0A277h

DWord	Bit	Description
0	31	**Render Power Meter Counter Overflow** Access: RO
	30:0	**Render PWRMTR Counter** Access: RO

Render Power Meter Counter No Clear

RPMCNTCLR - Render Power Meter Counter No Clear	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A28Ch-0A28Fh

Formerly cleared the count and the overflow bit, but now it is just a read-only value.

DWord	Bit	Description
0	31	**Render Power Meter Counter Overflow No Clear** Access: RO Formerly cleared the overflow bit, but now it is just a read-only value.
	30:0	**Render PWRMTR Counter No Clear** Access: RO Formerly cleared the count, but now is just a read-only value.

Render Promotion Timer - RC6

RC6TIMER - Render Promotion Timer - RC6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A0B8h-0A0BBh

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**RC6 Promote Time** Access: R/W Absolute time starting from post-hyst idle. 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_rc6_promotion_time[23:0]

RENDERRC0COUNTER

RENDERRC0COUNTER - RENDERRC0COUNTER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138118h

This register contains the total RC0 residency (Render powered on and clocks running) time that Render was in since boot.
SOXi Context Save/Restore : No
The 40-bit HW counter will wrap around. The only clear condition is CZ reset.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[4] controls if this register should count or if it should be gated: 0= clear, 1=count

DWord	Bit	Description
0	31:0	**RENDERRC0TIME** Default Value: 00000000h Access: RO Render RC0 Residency Counter.

RENDERRC1COUNTER

RENDERRC1COUNTER - RENDERRC1COUNTER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138110h

This register contains the total RC1 residency (Render powered on and clock gated) time that Render was in since boot.
SOXi Context Save/Restore : No
The 40-bit HW counter will wrap around. The only clear condition is CZ reset.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[2] controls if this register should count or if it should be gated: 0= clear, 1=count

DWord	Bit	Description
0	31:0	**RENDERRC1TIME** Default Value: 00000000h Access: RO Render RC1 Residency Counter.

RENDERRC6COUNTER

RENDERRC6COUNTER - RENDERRC6COUNTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138108h

This register contains the total RC6 residency (Render power gated and clock gated) time that Render power well was in since boot. The counter will wrap around.
SOXi Context Save/Restore : No
The time is given in units of CZ clock cycles. The counter will reset on CZ reset going high.
This means that a warm reset will also reset this counter and it will also wrap around when it reaches max with no indication that an overflow occurred.
This register will freeze the count value (stop counting, but not reset) when pmu_gvd_renwakeack_nczfwoh=1.
This register will count whenever pmu_gvd_renwakeack_nczfwoh=0.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.
0x13_8104[0] controls if this register should count or if it should be gated: 0= clear, 1=count.

DWord	Bit	Description
0	31:0	**RENDERRC6TIME** Default Value: 00000000h Access: RO Render Residency Counter.

Render TLB Control Register

RTCR - Render TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04260h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

Render Watchdog Counter

PR_CTR - Render Watchdog Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	02190h
Address: Name: ShortName:	12190h-12193h Watchdog Counter PR_CTR_VCSUNIT0
Address: Name: ShortName:	1A190h-1A193h Watchdog Counter PR_CTR_VECSUNIT
Address: Name: ShortName:	1C190h-1C193h Watchdog Counter PR_CTR_VCSUNIT1
Address: Name: ShortName:	22190h-22193h Watchdog Counter PR_CTR_BCSUNIT

DWord	Bit	Description
0	31:0	**Counter Value** Format: U32 This register reflects the render watchdog counter value itself. It cannot be written to.

Render Watchdog Counter Threshold

PR_CTR_THRSH - Render Watchdog Counter Threshold

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00150000
Access:	R/W
Size (in bits):	32
Address:	0217Ch
Address:	1217Ch-1217Fh
Name:	Watchdog Counter Threshold
ShortName:	PR_CTR_THRSH_VCSUNIT0
Address:	1A17Ch-1A17Fh
Name:	Watchdog Counter Threshold
ShortName:	PR_CTR_THRSH_VECSUNIT
Address:	1C17Ch-1C17Fh
Name:	Watchdog Counter Threshold
ShortName:	PR_CTR_THRSH_VCSUNIT1
Address:	2217Ch-2217Fh
Name:	Watchdog Counter Threshold
ShortName:	PR_CTR_THRSH_BCSUNIT

DWord	Bit	Description
0	31:0	**Counter logic Threshold** Default Value: 00150000h Format: U32 This field specifies the threshold that the hardware checks against for the value of the render clock counter before generating an interrupt. The counter in hardware generates an interrupt when the threshold is reached, rolls over and starts counting again. The interrupt generated is the "Media Hang Notify" interrupt since this watchdog timer is intended primarily to remedy VLD hangs on the main pipeline.

Reported BitRateControl parameter Mask

MFX_VP8_CNTRL_MASK - Reported BitRateControl parameter Mask

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12900h
Valid Projects:	CHV, BSW

This register stores the count of bytes of the bitstream output per frame

DWord	Bit	Description
0	31:6	**Reserved** Format: MBZ
	5	**Final Bitstream Buffer Overrun Mask** Format: U1 This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit5. This denotes Final bitstream buffer overrun feature is enabled.
	4	**Intermediate Bitstream Buffer Overrun Mask** Format: U1 This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit4. This denotes intermediate bitstream buffer overrun feature is enabled.
	3	**Intra MB Bit Count Conformance Mask** Format: U1 This is legacy support as AVC for Intra MB Bit Count conformance. This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit3. This feature is not validated.
	2	**Inter MB Bit Count Conformance Mask** Format: U1 This is legacy support as AVC for Inter MB Bit Count conformance. This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit2. This feature is not validated.

MFX_VP8_CNTRL_MASK - Reported BitRateControl parameter Mask

	1	**Frame Bit Rate Overflow Mask** Format: U1 This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit0. It denotes if Frame Bit Rate Overflow is enabled for Bit Rate Control
	0	**Frame Bit Rate Underflow Mask** Format: U1 This is same bit reflected in MFX_VP8_ENC_CFG DW2 bit1. It denotes if Frame Bit Rate Underflow is enabled for Bit Rate Control

Reported BitRateControl parameter Status

MFX_VP8_CNTRL_STATUS - Reported BitRateControl parameter Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12904h
Valid Projects:	CHV, BSW

This register stores the count of bytes of the bitstream output per frame

DWord	Bit	Description
0	31:8	**Reserved** Format: MBZ
	7	**QindexClampHigh Status** Format: U1 This denotes if Qindex is clamped by QindexClampHigh value programmed in MFX_VP8_PIC_STATE.DW7.
	6	**QindexClampLow Status** Format: U1 This denotes if Qindex is clamped by QindexClampLow value programmed in MFX_VP8_PIC_STATE.DW7.
	5	**Final Bitstream Buffer Overrun Status** Format: U1 This denotes if Final bitstream buffer overrun.
	4	**Intermediate Bitstream Buffer Overrun Status** Format: U1 This denotes if any of the Intermediate bitstream buffer overrun. (including FrameHeader, Partition1 to Partition8)
	3	**Intra MB Bit Count Conformance Status** Format: U1 This is legacy support as AVC for Intra MB Bit Count conformance. It denotes if Intra MB Bit Count meets conformance size. This feature is not validated.

MFX_VP8_CNTRL_STATUS - Reported BitRateControl parameter Status

	2	**Inter MB Bit Count Conformance Status** Format: U1 This is legacy support as AVC for Inter MB Bit Count conformance. It denotes if Inter MB Bit Count meets conformance size. This feature is not validated.
	1	**Frame Bit Rate Overflow Status** Format: U1 It denotes if Frame Bit Rate Overflow in current frame
	0	**Frame Bit Rate Underflow Status** Format: U1 It denotes if Frame Bit Rate Underflow in current frame

Reported Bitstream Output Bit Count for Syntax Elements Only Register

MFC_BITSTREAM_SE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128A4h
Valid Projects:	CHV, BSW

This register stores the count of number of bits in the bitstream due to syntax elements only. This excludes header/ byte alignment /tail/EMU/CABAC-0word/padding bits but includes the stop-one-bit. This register is part of the context save and restore.

DWord	Bit	Description
0	31:0	**MFC Bitstream Syntax Element Only Bit Count** Total number of bits in the bitstream output due to syntax elements only. It includes the data bytes only. This count is updated for every time the internal bitstream counter is incremented and its reset at image start.

Reported Bitstream Output Byte Count per Frame Register

MFC_BITSTREAM_BYTECOUNT_FRAME - Reported Bitstream Output Byte Count per Frame Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128A0h
Valid Projects:	CHV, BSW

This register stores the count of bytes of the bitstream output per frame

DWord	Bit	Description
0	31:0	**MFC Bitstream Byte Count per Frame** Total number of bytes in the bitstream output per frame from the encoder. This includes header/tail/byte alignment/data bytes/EMU (emulation) bytes/cabac-zero word insertion/padding insertion. This count is updated for every time the internal bitstream counter is incremented and its reset at image start.

Reported Bitstream Output CABAC Bin Count Register

MFC_AVC_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	128A8h
Valid Projects:	CHV, BSW

This register stores the count of number of bins per frame.

DWord	Bit	Description
0	31:0	**MFC AVC Cabac Bin Count** Total number of BINs in the bitstream output per frame from the encoder. This count is updated for every time the bin counter is incremented and its reset at image start.

Reported Final Bitstream Byte Count

MFX_VP8_FRM_BYTE_CNT - Reported Final Bitstream Byte Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	12908h
Valid Projects:	CHV, BSW

This register stores the count of bytes of the bitstream output per frame

DWord	Bit	Description
0	31:0	**Final BitStream Byte Count** Format: U32 This register contains Final Bitstream byte count

Reported Frame Zero Padding Byte Count

MFX_VP8_FRM_ZERO_PAD - Reported Frame Zero Padding Byte Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	1290Ch
Valid Projects:	CHV, BSW

This register stores Frame Zero Padding Byte Count

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Frame Zero Padding Byte Count** Format: U16 This register contains Frame Zero Padding byte count This is legacy support. This feature is not validated.

Reported Timestamp Count

TIMESTAMP - Reported Timestamp Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000h, 00000000 CHV, BSW
Access:	RO. This register is not set by the context restore.
Size (in bits):	64
Address:	02358h
Address:	12358h-1235Fh
Name:	Reported Timestamp Count
ShortName:	TIMESTAMP_VCSUNIT0
Address:	1A358h-1A35Fh
Name:	Reported Timestamp Count
ShortName:	TIMESTAMP_VECSUNIT
Address:	1C358h-1C35Fh
Name:	Reported Timestamp Count
ShortName:	TIMESTAMP_VCSUNIT1
Address:	22358h-2235Fh
Name:	Reported Timestamp Count
ShortName:	TIMESTAMP_BCSUNIT

Description

This register provides an elapsed real-time value that can be used as a timestamp for GPU events over short periods of time. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the PIPE_CONTROL command. See 3D Geometry Pipeline in the "3D and Media" volume. This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register is not reset by a graphics reset. It will maintain its value unless a full chipset reset is performed.

Note: On Core platforms, the TIMESTAMP register is initialized with the value of the PCU ART and hence tracks bits 38:3 of the 100 MHz ART fairly closely. However, due to variability in the actual time it takes to download the ART value to GT, the value of the TIMESTAMP register will be lower than the value of the PCU ART by an amount dependent on the relative IA/CLR/GT frequencies at the time the timestamp was downloaded to GT (expected to range between 100 and 600 ns). When comparing the value sampled from this register by GT HW to values read from the PCU timer by other system agents, timing differences between GT HW reading the TIMESTAMP register and the involved non-GT agent(s) reading the PCU ART must also be comprehended.

TIMESTAMP - Reported Timestamp Count

DWord	Bit	Description
0	63:36	**Reserved** Project: CHV, BSW Format: MBZ
	35:0	**Timestamp Value** Project: CHV, BSW Format: U36 **Description** This register toggles based on time stamp granularity (base unit) defined in the “Time Stamp Bases” subsection in Power Management chapter.

Reset Flow Control Messages

RSTFCTLMSG - Reset Flow Control Messages

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	08108h

Soft-Reset and FLR Flow Control Message Registers

DWord	Bit	Description
0	31:16	**Message Mask** Access: RO Message Mask In order to write to bits 15:0, the corresponding message mask bits must be written. For example, for bit 14 to be set, bit 30 needs to be 1 : 40004000
	15:12	**Reserved**
	11	**MEDIA 1 Reset flow acknowledgement message** Access: R/W PM Acknowledgement Messages for Media 1 reset: '1' : PREP_RST_MEDIA1_ACK - Acknowledgement that graphics media1 (or 2nd vbox) is prepared for reset assertion. '0' : DONE_MEDIA1_RST_ACK - Acknowledgement that graphics media1 (or 2nd vbox) reset is de-asserted
	10	**WIDI Reset flow acknowledgement message** Access: R/W PM Acknowledgement Messages for WIDI reset: '1' : PREP_RST_WIDI_ACK - Acknowledgement that graphics widi is prepared for reset assertion. '0' : DONE_WIDI_RST_ACK - Acknowledgement that graphics widi reset is de-asserted
	9	**Reserved**
	8	**Vebox Reset flow Acknowledge Message** Access: R/W PM Acknowledgement Messages for Vebox reset: '1' : PREP_RST_VEBOX_ACK - Acknowledgement that graphics VE is prepared for reset assertion. '0' : DONE_VEBOX_RST_ACK - Acknowledgement that graphics VE reset is de-asserted

RSTFCTLMSG - Reset Flow Control Messages

	7	**Blitter Reset Flow Acknowledgement Messages** Access: R/W PM Acknowledgement Messages for Blitter reset: '1' : PREP_RST_BLIT_ACK - Acknowledgement that graphics blitter is prepared for reset assertion. '0' : DONE_BLIT_RST_ACK - Acknowledgement that graphics blitter reset is de-asserted
	6	**Media Reset Flow Acknowledgement Messages** Access: R/W PM Acknowledgement Messages for Media reset: '1' : PREP_RST_MEDIA_ACK - Acknowledgement that graphics media block is prepared for reset assertion. '0' : DONE_MEDIA_RST_ACK - Acknowledgement that the graphics media reset is de-asserted
	5	**Render Reset Flow Acknowledgement Messages** Access: R/W PM Acknowledgement Messages for Render reset: '1' : PREP_RST_RENDER_ACK - Acknowledgement that the graphics render block is prepared for reset assertion. '0' : DONE_RENDER_RST_ACK - Acknowledgement that the graphics render reset is de-asserted
	4	**GTI-Device Reset Flow Acknowledgement Messages** Access: R/W PM Acknowledgement Messages for GTI-Device reset: '1' : PREP_RST_GTIDEV_ACK - Acknowledgement that the GTI device is prepared for reset assertion. '0' : DONE_GTIDEV_RST_ACK - Acknowledgement that the GTI device reset is de-asserted
	3	**Reserved** Access: RO Reserved
	2	**FLR Done ack from Pmunit** Access: R/W Set FLR Done ack from Pmunit: 1: PM unit sets this bit to acknowledge the FLR done message has been forwarded to SA through GAM interface. 0: Default Value. If the bit was set by PM then Cpunit hardware clears it once FLR is completed.

RSTFCTLMSG - Reset Flow Control Messages

	1	**Global Resource Arbitration Acknowledgement Messages** Access: R/W Global Resource Arbitration Acknowledgement Message from PM: '1' : CP_ARB_REQ_ACK - Acknowledgement for CPunit's global resource arbitration request '0' : CP_ARB_RELEASE_ACK - Acknowledgement to CPunit's release of global resources
	0	**CP Busy / Idle Status Acknowledgement Messages** Access: R/W CP Busy / Idle Status Acknowledgement Message from PM: '0' : CP_NOT_BUSY_ACK - Acknowledgement that the CPunit is idle. '1' : CP_BUSY_ACK - Acknowledgement that the CPunit is busy.

RESET Messaging Register for Clocking Unit

MSG_RESET_GCP - RESET Messaging Register for Clocking Unit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	16
Address:	08030h

Message registers have bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. In order to set bit0, for example, the data would be 0x0001_0001. In order to clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16. Message registers are protected from non-GT writes via the Message Channel.
Request to Prepare for Reset
1'b0 : Reset complete <default>
1'b1 : Prepare for reset

DWord	Bit	Description
0	15:8	**Reserved** Access: RO
	7	**Request to Prepare for FLR** Access: R/W [7] Prepare for devrst_b Domain Reset (FLR) Note: All resets except busrst_b will be asserted for an FLR
	6	**Request to Prepare for Media1 Reset** Access: R/W [6] Prepare for cmrst_b Domain Reset (vcs1unit)
	5	**Request to Prepare for Wi-Di Reset** Access: R/W [5] Prepare for cwrst_b Domain Reset (winunit)
	4	**Reserved**
	3	**Request to Prepare for Blitter Reset** Access: R/W [3] Prepare for crblitrst_b Domain Reset (bcsunit)
	2	**Request to Prepare for VEBox Reset** Access: R/W [2] Prepare for cvrst_b Domain Reset (vecsunit)

MSG_RESET_GCP - RESET Messaging Register for Clocking Unit

	1	**Request to Prepare for Media0 Reset** Access: R/W [1] Prepare for cmrst_b Domain Reset (vcs0unit)
	0	**Request to Prepare for Render Reset** Access: R/W [0] Prepare for crrst_b Domain Reset (csunit)

Resource Streamer Context Offset

RS_CXT_OFFSET - Resource Streamer Context Offset

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00003B00
Access:	Read/32 bit Write Only
Size (in bits):	32
Address:	021B4h

DWord	Bit	Description
0	31:6	**RS Offset** Format: U26 This field indicates the offset (64bytes granular) in to the logical rendering context to which Resource Streamer context is save/restored when enabled. This field register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle) and RC6 is disabled. On way to program this register is via Load Register Immediate command in the ring buffer as part of initialization sequence. Value: ECh — **[Default]** — DefaultValueDesc
	5:0	**Reserved** Format: MBZ

Resource Streamer Preemption Status

RS_PREEMPT_STATUS - Resource Streamer Preemption Status

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0215Ch

Preemption from First Level Batch Buffer: This register contains the offset in to the Batch Buffer on which Resource streamer got preempted. Note that it is offset from the Batch Start Address and not the graphics address corresponding to the preempted instruction on Batch Buffer. This register's contents will be looked at by Command Streamer on the next RS Start Trigger to provide the appropriate batch start address. Following preemption, context save happens on which this register is context saved, after context save the register gets auto reset by Command Streamer. This register gets context save/restored by Render Command Streamer as part of its render context. **Preemption from Second Level Batch Buffer:** This register contains the graphics address of the instruction in Second Level Batch Buffer on which Resource streamer got preempted. This register's contents will be looked at by Command Streamer on the next RS Start Trigger to provide the appropriate batch start address. Following preemption, context save happens on which this register is context saved, after context save the register gets auto reset by Command Streamer. This register gets context save/restored by Render Command Streamer as part of its render context.

Programming Notes

- This register is accessed by Render Command Streamer as part of render context save/restore; this register should be exercised by S/W only for resetting the register contents if required.
- Following preemption if there is no context save, SW should program this register with 0x0 so that it does not interfere with proceeding workloads.

DWord	Bit	Description
0	31:2	**Batch Buffer Offset** Format: Offset[31:2] This field specifies the DWord-aligned offset from the batch start address on which Resource Streamer got preempted.
	1	**RS_PREEMPT_STATUS** Format: MBZ This field when not set indicates RS got preempted on a natural sync point else it got preempted on a draw call.

RS_PREEMPT_STATUS - Resource Streamer Preemption Status

	0	**RS_PREEMPTED** Default Value: 0 Format: Enable If this bit is set indicates Resource Streamer got preempted. Other fields of this register are valid only when this bit is set.

Restored Timestamp LSDW

RTSLSDW - Restored Timestamp LSDW	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A530h-0A533h

DWord	Bit	Description
0	31:0	**Restored Time Stamp Storage** Access: R/W Restored Time Stamp Storage

Restored timestamp MSDW

RTSMSDW - Restored timestamp MSDW		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A534h-0A537h	
DWord	**Bit**	**Description**
0	31:0	**Restored Time Stamp Storage** Access: R/W Restored Time Stamp Storage

RID_CC

RID_CC - RID_CC

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	00008h

Revision Identification and Class code register. SOXi Context Save/Restore: Yes.

DWord	Bit	Description
0	31:24	**BASE_CLASS_CODE** Access: RO
	23:16	**SUB_CLASS_CODE** Access: RO When MGGC0[VAMEN] is 1, this value is 80h, indicating other multimedia device. When MGGC0[VAMEN] is 0 this value will be determined based on GGC register, GMS and IVD fields. When GGC[1] = 1 or GGC[7:3] = 5'b00000 this value is 80h, otherwise its 00h 00h: VGA compatible 80h: Non VGA (GMS = '00h' or IVD = '1b')
	15:8	**PROGRAMMING_INTERFACE** Default Value: 00h Access: RO
	7:0	**REVISION_ID** Default Value: 00000000b Access: R/W RID: The value in this field reflects the value of strapRID[7:0] (which is an input pin of GVD). For VV: The reset value will be the same as MID bits 23:16 (reference the MID register for MID register details). This register is read-able by any agent. Under SAI protection, the PMC and Punit (ie. 'Trusted_FW') can write this register and change the default value. No other agent has the ability to update this register. The PMC implementation will consider SRID and CRID before updating this register. Any PMC updates will be prior to Device 2 configuration. Note : The MID register will always reflect the stepping information. Even if PMC updates this register, the MID is available as a SRID reference.

RING_BUFFER_HEAD_PREEMPT_REG

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	0214Ch-0214Fh
Name:	RING_BUFFER_HEAD_PREEMPT_REG
ShortName:	RING_BUFFER_HEAD_PREEMPT_REG_RCSUNIT

Address:	1214Ch-1214Fh
Name:	RING_BUFFER_HEAD_PREEMPT_REG
ShortName:	RING_BUFFER_HEAD_PREEMPT_REG_VCSUNIT0

Address:	1A14Ch-1A14Fh
Name:	RING_BUFFER_HEAD_PREEMPT_REG
ShortName:	RING_BUFFER_HEAD_PREEMPT_REG_VECSUNIT

Address:	1C14Ch-1C14Fh
Name:	RING_BUFFER_HEAD_PREEMPT_REG
ShortName:	RING_BUFFER_HEAD_PREEMPT_REG_VCSUNIT1

Address:	2214Ch-2214Fh
Name:	RING_BUFFER_HEAD_PREEMPT_REG
ShortName:	RING_BUFFER_HEAD_PREEMPT_REG_BCSUNIT

Description

This register contains the Head pointer offset in the ring when the last PREEMPTABLE command was executed and caused the head pointer to move due to the UHPTR register being valid. If the PREEMPTABLE command is executed as part of the batch buffer then the value of the register will be the offset in the ring of the command past the batch buffer start that contained the preemptable command.
This is a global register and context save/restored as part of power context image.

Preemptable Commands	Source
• MI_ARB_CHECK • 3D_PRIMITIVE • GPGPU_WALKER • MEDIA_STATE_FLUSH • PIPE_CONTROL (Only in GPGPU mode of pipeline selection)	RenderCS

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

- MI_ATOMIC (Post Sync Operation set in GPGPU mode of pipeline selection)
- MI_SEMAPHORE_SIGNAL (Post Sync Operation set in GPGPU mode of pipeline selection)

Programming Notes

Programming Restriction:
This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:21	**Last Wrap Count**
	20:2	**Preempted Head Offset** Format: U19 This field contains the Head pointer offset in the ring when the last MI_ARB_CHECK command was executed and caused the head pointer to move due to the UHPTR register being valid.
	1:0	**Ring/Batch Indicator** Format: Enabled

Value	Name	Description
0h	Ring	Preemptable command was executed in ring and caused head pointer to be updated.
1h	Batch	Preemptable command was executed in batch and caused head pointer to be updated.
2h	2nd level batch	Preemptable command was executed in second level batch and caused head pointer to be updated.

Ring Buffer Control

RING_BUFFER_CTL - Ring Buffer Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	0203Ch-0203Fh
Name:	Ring Buffer Control
ShortName:	RING_BUFFER_CTL_RCSUNIT

Address:	1203Ch-1203Fh
Name:	Ring Buffer Control
ShortName:	RING_BUFFER_CTL_VCSUNIT0

Address:	1A03Ch-1A03Fh
Name:	Ring Buffer Control
ShortName:	RING_BUFFER_CTL_VECSUNIT

Address:	1C03Ch-1C03Fh
Name:	Ring Buffer Control
ShortName:	RING_BUFFER_CTL_VCSUNIT1

Address:	2203Ch-2203Fh
Name:	Ring Buffer Control
ShortName:	RING_BUFFER_CTL_BCSUNIT

Description	Source
These registers are used to define and operate the ring buffer mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the Programming Interface chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.	RenderCS
Ring Buffer Head and Tail Offsets must be properly programmed before it is enabled. A Ring Buffer can be enabled when empty.	BlitterCS, VideoCS, VideoEnhancementCS

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:12	**Buffer Length**

RING_BUFFER_CTL - Ring Buffer Control

Format:	U9-1 in 4 KB pages - 1

This field is written by SW to specify the length of the ring buffer in 4 KB Pages.Range = [0 = 1 page = 4 KB, 1FFh = 512 pages = 2 MB]

Value	Name	Description
0		1 page = 4 KB
1FFh		512 pages = 2 MB

11 **RBWait**

Description	Source
Indicates that this ring has executed a WAIT_FOR_EVENT instruction and is currently waiting. Software can write a "1" to clear this bit, write of "0" has no effect. When the RB is waiting for an event and this bit is cleared, the wait will be terminated and the RB will be returned to arbitration.	
RenderCS: RBWait is not set on executing WAIIT_FOR_EVENT instruction waiting on Async Flip Pending.	RenderCS

10 **Semaphore Wait**

Description
Indicates that this ring has executed a MI_SEMAPHORE_WAIT instruction and is currently waiting for wait condition to satisfy.

9 **Reserved**

Format:	MBZ

8 **Reserved**

Project:	CHV, BSW
Format:	MBZ

7:3 **Reserved**

Format:	MBZ

2:1 **Automatic Report Head Pointer**

Project:	CHV, BSW
Source:	PRM

Description
This field is written by software to control the automatic reporting (write) of this ring buffer's Head Pointer register (register DWord 1) to the corresponding location within the Hardware Status Page. Automatic reporting can either be disabled or enabled at 4KB, 64KB or 128KB boundaries within the ring buffer.
When Execlist Enable bit is set the head pointer will be reported to the head pointer location in the Per-Process Hardware Status Page. MI_AUTOREPORT_4KB option is not supported on A stepping.

RING_BUFFER_CTL - Ring Buffer Control

Value	Name	Description
0	MI_AUTOREPORT_OFF	Automatic reporting disabled
1	MI_AUTOREPORT_64KB	Report every 16 pages (64KB)
2	MI_AUTOREPORT_4KB	Report every page (4KB) This mode must not be enabled in Ring Buffer mode of scheduling to minimize the auto reports.
2	MI_AUTOREPORT_4KB	Report every page (4KB) This mode must not be enabled in Ring Buffer mode of scheduling to minimize the auto reports.
3	MI_AUTOREPORT_128KB	Report every 32 pages (128KB)

0 **Ring Buffer Enable**

Format:	Enable

This field is used to enable or disable this ring buffer. It can be enabled or disabled regardless of whether there are valid instructions pending. If disabled and the ring head equals ring tail, all state currently loaded in hardware is considered invalid.

Programming Notes	Source
Ring Buffer Mode of Scheduling: SW must follow the below programming notes during SW initialization phase or while enabling render engine's ring buffer for the first time, this would be coming out of boot, standby, hibernate or reset. SW must set the Force Wakeup bit to prevent GT from entering C6. • SW must dispatch workload (dummy context) to initialize render engine with default state such that any context switches that occur subsequently (Power Save) will save and restore coherent device state. Indirect pointers used in 3D states must point to valid graphics surface existing in memory. PP_DCLV followed by PP_DIR_BASE register should be programmed as part of initialization workload if PPGTT is enabled in GFX_MODE register. • SW must ensure all the register (MMIO) initialization/programming through CPU happens in this block or latter, this ensures the MMIO state is save/restored on subsequent context switches (Power Sequences). • Once the render engine is programmed with valid state and the configuration, Force Wakeup bit should be reset to enable C6 entry.	RenderCS
Render CS Only: Ring Buffer Mode of Scheduling: SW must follow the below programming notes before disabling ring buffer to ensure HW is not in middle of the IDLE flows. • SW must set the Force Wakeup bit to prevent GT from entering C6. • Disable IDLE messaging in CS (Write 0x2050[31:0] = 0x00010001) • Poll/Wait for register bits of 0x22AC[6:0] turn to 0x30 value. • Disable Ring Buffer • Enable IDLE messaging in CS (Write 0x2050[31:0] = 0x00010000) • Force Wakeup bit should be reset to enable C6 entry.	RenderCS

Ring Buffer Current Context ID Register

BCS_RCCID - Ring Buffer Current Context ID Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	BlitterCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	22190h-22197h

This register contains the current ring context ID associated with the ring buffer.

Programming Notes

The current context registers must not be written directly (via MMIO). The RCCID register should only be updated indirectly from RNCID.

DWord	Bit	Description
0	63:0	**Unnamed** See Context Descriptor for BCS.

Ring Buffer Head

RING_BUFFER_HEAD - Ring Buffer Head	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02034h-02037h
Name:	Ring Buffer Head
ShortName:	RING_BUFFER_HEAD_RCSUNIT
Address:	12034h-12037h
Name:	Ring Buffer Head
ShortName:	RING_BUFFER_HEAD_VCSUNIT0
Address:	1A034h-1A037h
Name:	Ring Buffer Head
ShortName:	RING_BUFFER_HEAD_VECSUNIT
Address:	1C034h-1C037h
Name:	Ring Buffer Head
ShortName:	RING_BUFFER_HEAD_VCSUNIT1
Address:	22034h-22037h
Name:	Ring Buffer Head
ShortName:	RING_BUFFER_HEAD_BCSUNIT

Description

This register is used to define and operate the ring buffer mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the Programming Interface chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions. **Ring Buffer Head Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.**

DWord	Bit	Description
0	31:21	**Wrap Count** Format: U11 count of ring buffer wraps This field is incremented by 1 whenever the **Head Offset** wraps from the end of the buffer back to the start (i.e., whenever it wraps back to 0). Appending this field to the **Head Offset** field effectively creates a virtual 4GB Head "Pointer" which can be used as a tag associated with instructions placed in a ring buffer. The Wrap Count itself will wrap to 0 upon overflow.

RING_BUFFER_HEAD - Ring Buffer Head

	20:2	**Head Offset** Format: GraphicsAddress[20:2] DWord Offset This field indicates the offset of the *next* instruction DWord to be parsed. Software will initialize this field to select the first DWord to be parsed once the RB is enabled. (Writing the Head Offset while the RB is enabled is UNDEFINED). Subsequently, the device will increment this offset as it executes instructions - until it reaches the QWord specified by the **Tail Offset**. At this point the ring buffer is considered "empty". **Programming Notes** A RB can be enabled empty or containing some number of valid instructions.
	1	**Reserved** Format: MBZ
	0	**Reserved** Project: CHV, BSW Format: MBZ

Ring Buffer Start

RING_BUFFER_START - Ring Buffer Start	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02038h-0203Bh
Name:	Ring Buffer Start
ShortName:	RING_BUFFER_START_RCSUNIT
Address:	12038h-1203Bh
Name:	Ring Buffer Start
ShortName:	RING_BUFFER_START_VCSUNIT0
Address:	1A038h-1A03Bh
Name:	Ring Buffer Start
ShortName:	RING_BUFFER_START_VECSUNIT
Address:	1C038h-1C03Bh
Name:	Ring Buffer Start
ShortName:	RING_BUFFER_START_VCSUNIT1
Address:	22038h-2203Bh
Name:	Ring Buffer Start
ShortName:	RING_BUFFER_START_BCSUNIT

Description

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a physical memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the Programming Interface chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions.

DWord	Bit	Description
0	31:12	**Starting Address** Format: GraphicsAddress[31:12]RingBuffer This field specifies Bits 31:12 of the 4KB-aligned starting Graphics Address of the ring buffer. Address bits 31 down to 29 must be zero. All ring buffer pages must map to Main Memory (uncached) pages. Ring Buffer addresses are always translated through the global GTT.
	11:0	**Reserved**

Ring Buffer Tail

RING_BUFFER_TAIL - Ring Buffer Tail

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02030h-02033h
Name:	Ring Buffer Tail
ShortName:	RING_BUFFER_TAIL_RCSUNIT
Address:	12030h-12033h
Name:	Ring Buffer Tail
ShortName:	RING_BUFFER_TAIL_VCSUNIT0
Address:	1A030h-1A033h
Name:	Ring Buffer Tail
ShortName:	RING_BUFFER_TAIL_VECSUNIT
Address:	1C030h-1C033h
Name:	Ring Buffer Tail
ShortName:	RING_BUFFER_TAIL_VCSUNIT1
Address:	22030h-22033h
Name:	Ring Buffer Tail
ShortName:	RING_BUFFER_TAIL_BCSUNIT

Description

These registers are used to define and operate the "ring buffer" mechanism which can be used to pass instructions to the command interface. The buffer itself is located in a linear memory region. The ring buffer is defined by a 4 Dword register set that includes starting address, length, head offset, tail offset, and control information. Refer to the Programming Interface chapter for a detailed description of the parameters specified in this ring buffer register set, restrictions on the placement of ring buffer memory, arbitration rules, and in how the ring buffer can be used to pass instructions. Ring Buffer Tail Offsets must be properly programmed before ring is enabled. A Ring Buffer can be enabled when empty.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ

RING_BUFFER_TAIL - Ring Buffer Tail

	20:3	**Tail Offset** Format: GraphicsAddress[20:3] This field is written by software to specify where the valid instructions placed in the ring buffer end. The value written points to the QWord past the last valid QWord of instructions. In other words, it can be defined as the next QWord that software will write instructions into. Software must write subsequent instructions to QWords following the Tail Offset, possibly wrapping around to the top of the buffer (i.e., software can't skip around within the buffer). Note that all DWords prior to the location indicated by the **Tail Offset** must contain valid instruction data - which may require instruction padding by software. See **Head Offset** for more information. **Programming Notes** [Ring Buffer Mode Of scheduling only][Video CS, Video Enhancement CS, Blitter CS]: HW loses Page Directory (PPGTT) information on becoming IDLE. SW must always program the PD information through MI_LOAD_REGISTER_IMM command in the ring buffer prior to programming workload begins on every ring dispatch. This will ensure Page Directory information is not lost due to IDLE flows.
	2:0	**Reserved** Format: MBZ

Root Table Address Pointer Value First 31_0

RTAPV_1_310 - Root Table Address Pointer Value First 31_0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F500h

This register is used to store local copy of the Root Table address pointer value.

DWord	Bit	Description
0	31:0	**First Address 31 to 0** Default Value: 00000000h Access: R/W Bits 31:11 = Root Table Address Pointer Value 31:11. Bits 10:1 = Reserved. Bit 0 = Enabled for Root Table Address Pointer Value 31:11.

Root Table Address Pointer Value Second 31_0

RTAPV_2_310 - Root Table Address Pointer Value Second 31_0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0F504h

This register is used to store local copy of the Root Table address pointer value.

DWord	Bit	Description
0	31:0	**Second Address 31 to 0** Default Value: 00000000h Access: R/W Bits 31:8 = Reserved. Bits 7:1 = Root Table Address Pointer Value 38:32. Bit 0 = Enabled for Root Table Address Pointer Value 38:32.

RP Decrease Limit

RPDECLIMIT - RP Decrease Limit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A030h-0A033h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Decrease Threshold** Access: R/W Decrease Threshold (DECLIMIT): This register contains the threshold used to determine whether a switch to a higher higher frequency is desirable. FRQDUM determines the meaning of this register: Busy Min Continuous Limit Busy Min Average Limit Idle Max Continuous Limit The values are: 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec

RP Down Timeout

RPDNTIMOUT - RP Down Timeout		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A010h-0A013h	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Access: RO
	23:0	**Down Timeout** Access: R/W The value in this register contains the timeout value for gfx idleness due to stopped graphics clocks (CPD) necessary to trigger the Render Geyserville Downward Timeout interrupt. 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec

RP Downwards Evaluation Interval

RPDNEI - RP Downwards Evaluation Interval

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A06Ch-0A06Fh

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Evaluation Interval Period for Downwards Freq Direction** Access: R/W The geyserville performance status will be averaged over this interval, and the results will be used to possibly recommend a switch to the slower render clock frequency (either directly by hardware or via an interrupt activating a sw decision). 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_ei_down[23:0]

RP Increase Limit

RPINCLIMIT - RP Increase Limit		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A02Ch-0A02Fh	

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Increase Threshold** Access: R/W Increase Threshold (INCLIMIT): This register contains the threshold used to determine whether a switch to a higher higher frequency is desirable. FRQIUM determines the meaning of this register: Busy Max Continuous Limit Busy Max Average Limit Idle Min Continuous Limit The values are: 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec

RP Normal Software Frequency Request

RP_FREQ_NORMAL - RP Normal Software Frequency Request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A008h

This 32 bit value is written to the PCU IO_THREAD_P_REQ register when NOT in Video Turbo mode.

DWord	Bit	Description
0	31	**Turbo Disable** Access: R/W The Turbo Disable bit is determined by SW. NOTE: If Turbo is disable for ANY thread, it will prevent turbo for ALL threads.
	30:24	**P State Request** Access: R/W This field indicates the maximum P-State request in units of 100MHz.
	23:18	**P State Offset** Access: R/W This field defined the number of steps that Energy Efficient P-State is allowed to fall in units of 100 MHz.
	17:14	**Energy Efficient policy** Access: R/W The energy efficiency policy is determined by SW.
	13:0	**Reserved** Access: RO

RP Software Frequency Request Hysteresis

RPSWFREQHYST - RP Software Frequency Request Hysteresis

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A004h

DWord	Bit	Description
0	31:6	**Reserved** Access: RO
	5:0	**RP SW Freq Request Hysteresis** Access: R/W Programming Notes: Not supported. It must be 0 at all times. — Project: CHV, BSW

RP Upwards Evaluation Interval

RPUPEI - RP Upwards Evaluation Interval

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A068h-0A06Bh

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**Evaluation Interval Period for Upwards Freq Direction** Access: R/W The geyserville performance status will be averaged over this interval, and the results will be used to possibly recommend a switch to the faster render clock frequency (either directly by hardware or via an interrupt activating a sw decision). 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_ei_up[23:0]

RP Video Turbo Software Frequency Request

RP_FREQ_VIDEOTURBO - RP Video Turbo Software Frequency Request

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A00Ch

This 32 bit value is written to the PCU IO_THREAD_P_REQ register when in Video Turbo mode.

DWord	Bit	Description
0	31	**Turbo Disable** Access: R/W The Turbo Disable bit is determined by SW. NOTE: If Turbo is disable for ANY thread, it will prevent turbo for ALL threads.
	30:24	**P State Request** Access: R/W This field indicates the maximum P-State request in units of 100MHz.
	23:18	**P State Offset** Access: R/W This field defined the number of steps that Energy Efficient P-State is allowed to fall in units of 100 MHz.
	17:14	**Energy Efficient policy** Access: R/W The energy efficiency policy is determined by SW.
	13:0	**Reserved** Access: RO

RS_PREEMPT_STATUS_UDW

RS_PREEMPT_STATUS_UDW - RS_PREEMPT_STATUS_UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02174h

Preemption from First Level Batch Buffer: This register contains the offset in to the Batch Buffer on which Resource streamer got preempted. Note that it is offset from the Batch Start Address and not the graphics address corresponding to the preempted instruction on Batch Buffer. This register's contents will be looked at by Command Streamer on the next RS Start Trigger to provide the appropriate batch start address. Following preemption, context save happens on which this register is context saved, after context save the register gets auto reset by Command Streamer. This register gets context save/restored by Render Command Streamer as part of its render context. **Preemption from Second Level Batch Buffer:** This register contains the graphics address of the instruction in Second Level Batch Buffer on which Resource streamer got preempted. This register's contents will be looked at by Command Streamer on the next RS Start Trigger to provide the appropriate batch start address. Following preemption, context save happens on which this register is context saved, after context save the register gets auto reset by Command Streamer. This register gets context save/restored by Render Command Streamer as part of its render context.

Programming Notes

- This register is accessed by Render Command Streamer as part of render context save/restore; this register should be exercised by S/W only for resetting the register contents if required.
- Following preemption if there is no context save, SW should program this register with 0x0 so that it does not interfere with proceeding workloads.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Offset Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted second level batch buffer in resource streamer.

RS Preemption Hint

RS_PRE_HINT - RS Preemption Hint

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	024C0h

This register contains the Dword aligned Graphics address in to the Batch Buffer corresponding to either 3D_PRIMITIVE or RCS-RS sync command called Preemption Hint Address. When Preemption Hint Address is enabled, RS will honor preemption request from RCS only on parsing 3D_PRIMITIVE/RCS-RS sync command at Preemption Hint Address.

Programming Notes

Programming Restriction:
This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of preempting Resource Streamer in command stream. Programmer has to ensure that RS Preemption Hint register gets programmed well before RS gets preempted by RCS. Note that this register should be programmed with caution as it can lead to indefinite stalls in RS.

a. RS will preempt on receiving preemption request from RCS only on reaching the instruction in the batch buffer corresponding to the address mentioned in RS_PREEMPT_HINT. RS could hit an RCS-RS sync command before reaching the address mentioned in the RS_PREEMPT_HINT, in this case RS should preempt on the sync command.

b. RS could hit the address mentioned in 3D_PREEMPT_HINT before receiving preempt request from RCS. In this case RS will stall at this command until it receives preemption request from RCS and then preempts.

DWord	Bit	Description
0	31:2	**Preemption Hint Address** Format: U30 This field contains the Dword aligned Graphics Address in to the batch buffer as Preemption Hint.
	1	**Reserved**
	0	**Preemption Hint** Format: Enabled 0h – Disabled – Preemption hint is disabled for Resource Streamer. 1h – Enabled – Preemption hint is enabled for Resource streamer.

RS Preemption Hint UDW

RS_PREEMPTION_HINT_UDW - RS Preemption Hint UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	024C4h

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the Batch Buffer corresponding to either 3D_PRIMITIVE or RCS-RS sync command called Preemption Hint Address. When Preemption Hint Address is enabled, RS will honor preemption request from RCS only on parsing 3D_PRIMITIVE/RCS-RS sync command at Preemption Hint Address.

Restriction

This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of preempting Resource Streamer in command stream. See RS_PRE_HINT definition for further restrictions.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Preemption Hint Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer as Preemption Hint.

Sampler control register

SAMPLER_CTL - Sampler control register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0E140h
Valid Projects:	CHV, BSW]

DWord	Bit	Description
0	31:16	**ECO Reserved 1** Reserved: MBZ
	15:8	**Reserved**
	7:3	**Sampler unit select**
	2	**ECO Reserved 2**
	1:0	**ECO Reserved 3**

SAMPLER Mode Register

SAMPLER_MODE - SAMPLER Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Size (in bits):	32
Trusted Type:	1
Address:	07028h
Valid Projects:	CHV, BSW

This register has bit-wise masking applied for writes. The register consists of 16 bits of data in [15:0], and 16 bits of corresponding masks in [31:16]. To set bit0, for example, the data would be 0x0001_0001. To clear bit0, for example, the data would be 0x0001_0000. Note that mask bit is the data bit offset + 16.

DWord	Bit	Description
0	31:16	**Reserved** Access: RO
	15:14	**ECO Reserved 1** Format: MBZ
	13:8	**ECO Reserved 2** Project: CHV, BSW Format: MBZ
	7:6	**ECO Reserved 3** Project: All Format: MBZ
	5	**ECO_SCRATCH3B** Project: Format: MBZ
	4:0	**Sample_d Quality Mode** Project: CHV, BSW Format: U5 This field configures the image quality mode for the sample_d message in the sampling engine. In general, performance will increase with each step of reduced quality.

Value	Name	Description	Project
00h	Disabled	Full quality is enabled, matching prior products	All
01h-1Fh		Quality degrades with each larger value, performance improves with each larger value	All

SAMPLER READ DATA

SAMPLER_RDATA - SAMPLER READ DATA

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0xFFFFFFFF
Access:	RO Variant
Size (in bits):	32
Address:	0E144h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:0	**Reserved** Default Value: FFFFFFFFh

Save Timer

SVTIMER - Save Timer

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x60001000
Size (in bits):	32
Address:	0B434h

DWord	Bit	Description
0	31	**Reserved**
	30:29	**Counter Enabling Selection** Default Value: 11b Access: R/W LPFC provides rudimentary compression by allowing software to select from several predefined levels of event reporting. Based on the value of this bitfield, only a certain number of the programmed events in the "Event Selection and Base Counters" registers (CNT0CL, CNT1CL, ..., CNT7CL) will be tracked and reported: Value / Selected Counters 00 - Counter 0 01 - Counters 0 & 1 10 - Counters 0, 1, 2, & 3 11 - Counters 0 - 7 Signal - lpconf_lpfc_cnt_enabled [1:0].
	28:24	**Reserved**
	23:0	**Save Timer Intervel** Default Value: 000000000001000000000000b Access: R/W Save Timer Interval (SVTMRINT). Save Timer Interval: This is the interval for sampling the performance counters and writing to memory. Each time it expires, the counters are sampled and packetized to be sent to DMA controller. The minimum granularity of sampling period if 256clocks. The value in this register is used as 256 x value to find the sampling window. For a 1Ghz core clock it provides up to 4ns of sampling period while matching the maximum capability of the event counters. 1h - 256clks. 2h - 512clks. ... 8h - 2048clks. Signal - lpconf_lpfc_savetimer_int [23:0].

SB_ADDRESS

SB_ADDRESS - SB_ADDRESS

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182108h

The Sideband Address Register is used by the sideband transaction (triggered by Sideband Packet Register) for holding the address (of the internal register, within the destination unit).

IOSF SB access prevention: An IOSF SB write access targeting this register will complete with no affect.
An IOSF SB read access targeting this register will abort (with 1's being returned on IOSF SB).

DWord	Bit	Description
0	31:0	**SB_Addr** Default Value: 00000000h Access: R/W Address, written by IOSF primary, for triggered IOSF SB initiated access. 0x18_2100 (SB_Busy) bit 0 =1 then an IOSF primary write will NOT be captured. 0x18_2100 (SB_Busy) bit 0 = 0 then an IOSF primary write will be captured. Dword aligned addresses must be used.

SB_DATA

SB_DATA - SB_DATA

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182104h

The Sideband Data Register is used by the sideband register access mechanism (triggered by Sideband Packet Register) for holding write-data in write-transactions or read-data for read-transactions as explained below.

IOSF SB access prevention :An IOSF SB write access targeting this register will complete with no affect.
An IOSF SB read access targeting this register will abort (with 1's being returned on IOSF SB).
Requirement : IOSF primary write to this register should not occur when a pending IOSF SB return is pending.

DWord	Bit	Description
0	31:0	**SB Data** Default Value: 00000000h Access: R/W Triggered IOSF SB write, this is the Data written in the IOSF SB initiated access. Triggered IOSF SB read, this is the Data read return from the IOSF SB initiated read access. 0x18_2100 (SB_Busy) bit 0 = 1 then an IOSF primary write will NOT be captured. 0x18_2100 (SB_Busy) bit 0 = 0 then an IOSF primary write will be captured.

SB_REQ_TRIGGER

SB_REQ_TRIGGER - SB_REQ_TRIGGER	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	182100h

When the Sideband Packet Register is written, the Gunit creates a transaction towards the destination agent on the IOSF sideband channel.
The fields sent with the write operation as transaction parameters:

- The Sideband Rid field (bits 31:24) is used as the transaction Rid.
- The Sideband Opcode field (bits 23:15) is used as the transaction opcode.
- The Sideband Port field (bits 15:8) is used as the transaction destination port.
- The source port is hard-coded to 6h (Gunit port).
- The Sideband Byte Enable Field (bits 7:4) is used as the transaction Byte Enable.

If the opcode results in a data write semantic transaction, the write-data will be taken from the Sideband Data Register. If the opcode results in a data read semantic transaction, the read-data will be placed in the Sideband Data Register and may later be read by software. When Sideband Busy is set, Sideband Packet Register, Sideband Data Register and Sideband Address Register fields cannot be written. If the opcode results in a data read semantic transaction, data will be ready at Sideband Data register only when the Sideband Busy is cleared. Fuse block access prevention:To prevent an attacker from using this mechanism to read the fuse block, writing the Fuse Block PortID to the Sideband Port will prevent the SB_busy bit from being set. This effectively 'aborts' the access. The triggered write is effectively dropped. The triggered read would supply whatever happened to be in the IOSF Sideband Doorbell Data register. IOSF SB access prevention :No usage models require IOSF SB accesses to the doorbell registers. No IOSF SB sources should be using IOSF SB accesses to the doorbell register to trigger a doorbell generated IOSF SB message. Software and firmware are PROHIBITED from using IOSF SB accesses to target and trigger accesses from this registers.

Usage of the doorbell mechanism :
From IOSF primary, to initiate a write on IOSF Sideband :
a. Write Data (0x18_2104)
b. Write Address (0x18_2108)
c. Write Packet and Trigger (0x18_2100)
d. Read poll 0x18_2100. When bit 0 = '0', then the write has completed on IOSF Sideband.
From IOSF primary, to initiate a read on IOSF Sideband :
a. Write Address (0x18_2108)
b. Write Packet and Trigger (0x18_2100)
c. Read poll 0x18_2100. When bit 0 = '0', then the read has completed on IOSF Sideband and data is available.
d. Read IOSF SB returned Data (0x18_2104)
If software attempts a read to a register and the result is a UR (Unsupported Request), the busy bit will be cleared with no update of the doorbell data register.

SB_REQ_TRIGGER - SB_REQ_TRIGGER

DWord	Bit	Description
0	31:24	**SB_DevFn** Default Value: 00h Access: R/W Device and Function number to be used for the IOSF SB access. Per the IOSF Specification, this 8-bit field ('fid' in the IOSF Spec) is a unique identifier of a target in a hierarchy of PCI buses (indicates Device Number [31:27] / Function Number [26:24] or Function Number [31:24]). The target is free to utilize this in an agent-specific manner.
	23:16	**SB_Opcode** Default Value: 00h Access: R/W Opcode to be used for the IOSF SB access.
	15:8	**SB_Port** Default Value: 00h Access: R/W Port to be used for the IOSF SB access.
	7:4	**SB_ByteEnables** Default Value: 0h Access: R/W Byte enables to be used for the iOSF SB access.
	3:1	**SB_BAR** Default Value: 000b Access: R/W BAR value
	0	**SB_Busy** Default Value: 0b Access: RO A write to this register will set this bit = '1' and triggers a IOSF SB request. When this bit is '1', the following registers are not IOSF primary write-able (0x18_2100, 0x18_2104, 0x18_2108)The completion of the IOSF SB access will clear this bit. 0 - 0x18_2100, 0x18_2104 and 0x18_2108 are writeable via IOSF Sideband. 1 - An IOSF SB access is in-progress. 0x18_2100, 0x18_2104 and 0x18_2108 are NOT writeable via IOSF Sideband.

SCPD0

SCPD0 - SCPD0			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	18209Ch		
Scratch Pad 0 Register			
DWord	**Bit**	**Description**	
0	31:0	**Scratch Pad**	
		Default Value:	00000000h
		Access:	R/W
		Software scratch pad	

SCRATCH1

SCRATCH1 - SCRATCH1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 CHV, BSW
Size (in bits):	32
Address:	0B11Ch

DWord	Bit	Description
0 **Project:** CHV, BSW	31:0	**SCRATCH** Project: CHV, BSW Access: R/W

SCRATCH for LNCFunit

SCRATCH_LNCF1 - SCRATCH for LNCFunit

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000001 CHV, BSW
Size (in bits):	32
Address:	0B008h

DWord	Bit	Description
0	31:3	**SCRATCH register for LNCFunit** Project: CHV, BSW Access: R/W
	2	**Memory fill delay** Access: R/W lncf_csr_lni_gt2_memfill_dis. 0:mem fills gt2 latency will be 1 . 1:mem fill gt2 latency will be same as gt3.
	1	**flush start delay** Access: R/W lncf_csr_lni_disable_flush_start_delay. 0:Flush processing in LNIunit starts one clock after receiving the flush command default. 1:Flush processing in LNIunit starts in the same clock in which flush command is received.
	0	**Non-IA coherent atomics enable** Default Value: 1b Project: CHV, BSW Access: R/W 0: atomics in GTI (). 1: atomics in L3 (non-IA atomic) (Default). Output signal from LNCF unit lncf_csr_lni_glblatmcs_l3. Value for this bit should be same as lbcf_csr_lsqc_glblatmcs_l3 b118[22].Value of this bit should be same as LBCF register bit 0xb11c[8].Adding Xbuf 8 MCP.

Scratch Register 1

SCRATCH1 - Scratch Register 1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A188h-0A18Bh	
DWord	**Bit**	**Description**
0	31:0	**Scratch1** Access: R/W Register bits that have no connection to design. Used to enable/disable changes to the design that were put in during the ECO process. none

Scratch Register 2

SCRATCH2 - Scratch Register 2		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A198h	
DWord	**Bit**	**Description**
0	31:0	**Scratch2** Access: R/W Register bits that have no connection to design. Used to enable/disable changes to the design that were put in during the ECO process. none

Second Buffer Size

SBS - Second Buffer Size		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0B424h	
DWord	**Bit**	**Description**
0	31:16	**Second Virtual Buffer Base** Access: R/W Second Virtual Buffer Base (SVBB0). Second Virtual Buffer Base: Programmed by driver to allocate a memory space for performance data storage. The buffer size should be aligned to the size of the memory allocated so it naturally aligns to the base (i.e. for 128KB bit[16]=0, 256KB bit[17:16]=0, 512KB bit[18:16]=0). Signal - lpconf_lpfc_virtual_base1 [31:16].
	15:12	**Second Buffer Size 0** Access: R/W Second Buffer Size: Determines the allowed buffer size for performance data storage. 0000b: 64KB. 0001b: 128KB. 0010b: 256KB. 0011b: 512KB. ... 1111b: 2GB. Signal - lpconf_lpfc_buffer_size1 [3:0].
	11:0	**Reserved** Access: RO Reserved.

Second Level Batch Buffer Head Pointer Preemption Register

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	0213Ch-0213Fh
Name:	Second Level Batch Buffer Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_RCSUNIT
Address:	1213Ch-1213Fh
Name:	Second Level Batch Buffer Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_VCSUNIT0
Address:	1A13Ch-1A13Fh
Name:	Second Level Batch Buffer Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_VECSUNIT
Address:	1C13Ch-1C13Fh
Name:	Second Level Batch Buffer Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_VCSUNIT1
Address:	2213Ch-2213Fh
Name:	Second Level Batch Buffer Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_BCSUNIT

Description

This register gets updated with the DWord-aligned graphics memory address of the PREEMPTABLE command in the second level batch buffer on which preemption has occurred.

This register value should be looked at only when the preemption has occurred in the second level batch buffer. This is indicated by "Ring/Batch Indicator" in "RING_BUFFER_HEAD_PREEMPT_REG". This register value retains its previous value and doesn't change when the preemption occurs on a preemptable command in ring buffer or in batch buffer.

Preemption is triggered by valid UHPTR in ring buffer mode of scheduling and by a pending execlist in Exec-List mode of scheduling. This is a global register and context save/restored as part of power context image.

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

Preemptable Commands	Source
MI_ARB_CHECK 3D_PRIMITIVE GPGPU_WALKER MEDIA_STATE_FLUSH PIPE_CONTROL (Only in GPGPU mode of pipeline selection) MI_ATOMIC (Post Sync Operation set in GPGPU mode of pipeline selection) MI_SEMAPHORE_SIGNAL (Post Sync Operation set in GPGPU mode of pipeline selection)	RenderCS

Programming Notes
Programming Restriction: This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:2	**Second Level Batch Buffer Head Pointer** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address of the PREEMPTABLE command in a batch buffer where the Preemption has occurred.
	1:0	**Reserved** Format: MBZ

Second Level Batch Buffer Head Pointer Register

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	CommandStreamer
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	02114h-02117h
Name:	Second Level Batch Buffer Head Pointer Register
ShortName:	SBB_ADDR_RCSUNIT
Address:	12114h-12117h
Name:	Second Level Batch Buffer Head Pointer Register
ShortName:	SBB_ADDR_VCSUNIT0
Address:	1A114h-1A117h
Name:	Second Level Batch Buffer Head Pointer Register
ShortName:	SBB_ADDR_VECSUNIT
Address:	1C114h-1C117h
Name:	Second Level Batch Buffer Head Pointer Register
ShortName:	SBB_ADDR_VCSUNIT1
Address:	22114h-22117h
Name:	Second Level Batch Buffer Head Pointer Register
ShortName:	SBB_ADDR_BCSUNIT

This register contains the current DWord Graphics Memory Address of the last-initiated batch buffer.

Programming Notes

This register should NEVER be programmed by driver, this is for HW internal use only. This register should not be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer. This register is saved and restored with context.

DWord	Bit	Description
0	31:2	**Second Level Batch Buffer Head Pointer** Format: GraphicsAddress[31:2] This field specifies the DWord-aligned Graphics Memory Address where the last initiated Second Level Batch Buffer is currently fetching commands. This field is meaningful only when Valid field is set to "1".
	1	**Reserved** Format: MBZ

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

	0	**Valid** Format: U1 **Value** / **Name** / **Description** / **Project** 0h / Invalid **[Default]** / Second Level Batch buffer Invalid / CHV, BSW 1h / Valid / Second Batch buffer Valid. / CHV, BSW

Second Level Batch Buffer State Register

SBB_STATE - Second Level Batch Buffer State Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	02118h
Address:	12118h-1211Bh
Name:	Second Level Batch Buffer State Register
ShortName:	SBB_STATE_VCSUNIT0
Address:	1A118h-1A11Bh
Name:	Second Level Batch Buffer State Register
ShortName:	SBB_STATE_VECSUNIT
Address:	1C118h-1C11Bh
Name:	Second Level Batch Buffer State Register
ShortName:	SBB_STATE_VCSUNIT1
Address:	22118h-2211Bh
Name:	Second Level Batch Buffer State Register
ShortName:	SBB_STATE_BCSUNIT

This register contains the attributes of the second level batch buffer initiated from the batch Buffer.

This register should not be written by software. These fields should only get written by a context restore. Software should always set these fields via the MI_BATCH_BUFFER_START command when initiating a batch buffer. This register is saved and restored with context.

DWord	Bit	Description
0	31:9	**Reserved** Format: MBZ
	8	**Reserved** Project: CHV, BSW Format: MBZ
	7	**Resource Streamer Enable** Format: U1 When this bit is set, the Resource Streamer will execute the batch buffer. When this bit is clear the Resource Streamer will not execute the batch buffer.

SBB_STATE - Second Level Batch Buffer State Register

	6	**Reserved** Project: CHV, BSW Format: MBZ
	5	**Address Space Indicator** Project: CHV, BSW Note: This field reflects the effective address space indicator security level and may not be the same as the Address Space Indicator written using MI_BATCH_BUFFER_START.

Value	Name	Description
0h	GGTT **[Default]**	This second level batch buffer is located in GGTT memory and is privileged
1h	PPGTT	This second level batch buffer is located in PPGTT memory and is non-privileged.

	4	**Reserved** Project: CHV, BSW Format: MBZ
	3:0	**Reserved** Format: MBZ

Second Level Batch Buffer Upper Head Pointer Preemption Register

SBB_PREEMPT_ADDR_UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	02138h-0213Bh
Name:	Second Level Batch Buffer Upper Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_UDW_RCSUNIT

Address:	12138h-1213Bh
Name:	Second Level Batch Buffer Upper Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_UDW_VCSUNIT0

Address:	1A138h-1A13Bh
Name:	Second Level Batch Buffer Upper Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_UDW_VECSUNIT

Address:	1C138h-1C13Bh
Name:	Second Level Batch Buffer Upper Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_UDW_VCSUNIT1

Address:	22138h-2213Bh
Name:	Second Level Batch Buffer Upper Head Pointer Preemption Register
ShortName:	SBB_PREEMPT_ADDR_UDW_BCSUNIT

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the last preempted second level batch buffer. This register follows the same rules as the SBB_PREEMPT_ADDR register.

Programming Notes

Programming Restriction: This register should NEVER be programmed by driver, this is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved**
	15:0	**Second Level Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space of the last preempted second level batch buffer.

Second Level Batch Buffer Upper Head Pointer Register

SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32

Address:	0211Ch-0211Fh
Name:	Second Level Batch Buffer Upper Head Pointer Register
ShortName:	SBB_ADDR_UDW_RCSUNIT

Address:	1211Ch-1211Fh
Name:	Second Level Batch Buffer Upper Head Pointer Register
ShortName:	SBB_ADDR_UDW_VCSUNIT0

Address:	1A11Ch-1A11Fh
Name:	Second Level Batch Buffer Upper Head Pointer Register
ShortName:	SBB_ADDR_UDW_VECSUNIT

Address:	1C11Ch-1C11Fh
Name:	Second Level Batch Buffer Upper Head Pointer Register
ShortName:	SBB_ADDR_UDW_VCSUNIT1

Address:	2211Ch-2211Fh
Name:	Second Level Batch Buffer Upper Head Pointer Register
ShortName:	SBB_ADDR_UDW_BCSUNIT

This register contains the current Upper DWord of Graphics Memory Address of the last-initiated batch buffer.
Programming Restriction:
This register should NEVER be programmed by driver. This is for HW internal use only.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Batch Buffer Head Pointer Upper DWORD** Format: GraphicsAddress[47:32] This field specifies the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space where the last initiated Batch Buffer is currently fetching commands. If no batch buffer is currently active, the Valid bit in BB_ADDR will be 0 and this field is meaningless.

Semaphore Polling Interval on Wait

SEMA_WAIT_POLL - Semaphore Polling Interval on Wait	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	0224Ch
Address:	1224Ch-1224Fh
Name:	Semaphore Polling Interval on Wait
ShortName:	SEMA_WAIT_POLL_VCSUNIT0
Address:	1A24Ch-1A24Fh
Name:	Semaphore Polling Interval on Wait
ShortName:	SEMA_WAIT_POLL_VECSUNIT
Address:	1C24Ch-1C24Fh
Name:	Semaphore Polling Interval on Wait
ShortName:	SEMA_WAIT_POLL_VCSUNIT1
Address:	2224Ch-2224Fh
Name:	Semaphore Polling Interval on Wait
ShortName:	SEMA_WAIT_POLL_BCSUNIT

The SEMA_WAIT_POLL register contains Poll Interval field which specifies the minimum number of microseconds allowed for command streamer to wait before re-fetching the data from the address mentioned in the MI_SEMAPHORE_WAIT command on WAIT Mode set to POLL until the condition is satisfied while the context is not switched out.
When a value of 0 is written the poll interval will be equal to the memory latency of the read completion.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:0	**Poll Interval** Minimum number of micro-seconds allowed

SSID_SID

SSID_SID - SSID_SID

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0002Ch

This register is used to uniquely identify the subsystem where the PCI device resides.

DWord	Bit	Description
0	31:16	**SUBID** Default Value: 0000h Access: R/W Once This value is used to identify the vendor of the subsystem.This register is programmed by BIOS during boot-up.Once written, this register becomes Read_Only. This register is cleared by a Reset.
	15:0	**SUBVID** Default Value: 0000h Access: R/W Once This value is used to identify the vendor of the subsystem.This register is programmed by BIOS during boot-up.Once written, this register becomes Read_Only. This register is cleared by a Reset.

Staggered EU/SAMPLER PAUSE on Frequency Change

GFXPAUSE - Staggered EU/SAMPLER PAUSE on Frequency Change

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A000h-0A003h

DWord	Bit	Description
0	31:19	**Reserved** Access: RO Reserved
	18	**Pause Lock** Access: R/W Lock 0 = Bits of EUPAUSE are R/W 1 = All bits of EUPAUSE are RO (including this lock bit)
	17	**EU Pause Enable** Access: R/W Lock 0 = Disabled; EUs will not be paused during frequency changes 1 = Enabled; EUs will be paused before graphics clocks are gated, and unpaused (staggered per EU) when the clocks are ungated
	16	**Sampler Stall Enable** Access: R/W Lock 0 = Disabled; Sampler will not be paused during frequency changes 1 = Enabled; Sampler will be paused before graphics clocks are gated, and unpaused when the clocks are ungated
	15:0	**Pause Count** Access: R/W Lock This is the minimum time the PMunit waits after asserting the EU or Sampler pause (if those are enabled) before allowing the core clocks to be gated. 0000 = Disabled 0001 - Count 1 CSclk ... FFFF = Count 65535 CSclks

Storage 1

STORAGE1 - Storage 1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A500h-0A503h	
DWord	**Bit**	**Description**
0	31:16	**Wake Rate Counter Render** Access: R/W When reading, this field holds the number of times that the render well has awakened from RC1 (i.e. the number of times that (pmmr_cs_done & ~gmcrgu_gpm_renderpower_req) changes from 1 to 0) within an evaluation interval. When writing this register, set the render wakerate counter to the value written. The only reason to read this register is preparation for S0ix, though it may be useful for validation. The only reason to write to this register is to resume from S0ix.
	15:0	**Wake Rate Counter Media** Access: R/W When reading, this field holds the number of times that the media well has awakened from RC1 (i.e. the number of times that (pmmr_vcs_done & pmmr_bcs_done & ~gmcrgu_gpm_mediapower_req) changes from 1 to 0) within an evaluation interval. When writing this register, set the render wakerate counter to the value written. The only reason to read this register is preparation for S0ix, though it may be useful for validation. The only reason to write to this register is to resume from S0ix.

Storage 2

STORAGE2 - Storage 2			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0A504h-0A507h		
DWord	**Bit**	**Description**	
0	31:24	**Reserved**	
		Access:	RO
	23:0	**RC EI Counter Media**	
		Access:	R/W

Storage 3

STORAGE3 - Storage 3		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A508h-0A50Bh	

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**RC EI Counter Render** Access: R/W

Storage 4

STORAGE4 - Storage 4			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0A50Ch-0A50Fh		
DWord	**Bit**	**Description**	
0	31:24	**Reserved**	
		Access:	RO
	23:0	**RC Idle Counter Media**	
		Access:	R/W

Storage 5

STORAGE5 - Storage 5			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0A510h-0A513h		
DWord	**Bit**	**Description**	
0	31:24	**Reserved**	
		Access:	RO
	23:0	**RC Idle Counter Render**	
		Access:	R/W

Storage 6

STORAGE6 - Storage 6			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0A514h-0A517h		
DWord	**Bit**	**Description**	
0	31:24	**Reserved**	
		Access:	RO
	23:0	**RP EI Up Counter**	
		Access:	R/W

Storage 7

STORAGE7 - Storage 7		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	0A518h-0A51Bh	

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**RP EI Up Busy Counter** Access: R/W

Storage 8

STORAGE8 - Storage 8			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	PRM		
Default Value:	0x00000000		
Size (in bits):	32		
Address:	0A51Ch-0A51Fh		
DWord	**Bit**	**Description**	
0	31:24	**Reserved**	
		Access:	RO
	23:0	**RP EI Down Counter**	
		Access:	R/W

Storage 9

STORAGE9 - Storage 9	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A520h-0A523h

DWord	Bit	Description
0	31:24	**Reserved** Access: RO
	23:0	**RP EI Down Busy Counter** Access: R/W

Stream Output Num Primitives Written Counter

SO_NUM_PRIMS_WRITTEN[0:3] - Stream Output Num Primitives Written Counter

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	05200h-0521Fh

There is one 64-bit register for each of the 4 supported streams:5200h-5207h SO_NUM_PRIMS_WRITTEN0 (for Stream Out Stream #0)5208h-520Fh SO_NUM_PRIMS_WRITTEN1 (for Stream Out Stream #1)5210h-5217h SO_NUM_PRIMS_WRITTEN2 (for Stream Out Stream #2)5218h-521Fh SO_NUM_PRIMS_WRITTEN3 (for Stream Out Stream #3)These registers are used to count the number of primitives (aka objects: points, lines, triangles) which the SO stage has successfully written to a particular "stream's" Streamed Vertex Output buffers, subject to buffer overflow detection. (See the Stream Output section of the 3D pipeline volume).These registers are part of the context save and restore.

DWord	Bit	Description
0	63:0	**Num Prims Written Count** Format: U64 This count is incremented (by one) every time a GS thread outputs a DataPort Streamed Vertex Buffer Write message with the Increment Num Prims Written bit set in the message header (see the Geometry Shader and Data Port chapters in the 3D Volume.)

Stream Output Primitive Storage Needed Counters

SO_PRIM_STORAGE_NEEDED[0:3] - Stream Output Primitive Storage Needed Counters

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	RW. This register is set by the context restore.
Size (in bits):	64
Address:	05240h-0525Fh

There is one 64-bit register for each of the 4 supported streams:

5240h-5247h SO_PRIM_STORAGE_NEEDED0 (for Stream Out Stream #0)
5248h-524Fh SO_PRIM_STORAGE_NEEDED1 (for Stream Out Stream #1)
5250h-5257h SO_PRIM_STORAGE_NEEDED2 (for Stream Out Stream #2)
5258h-525Fh SO_PRIM_STORAGE_NEEDED3 (for Stream Out Stream #3)

These registers are used to count the number of primitives (aka objects: points, lines, triangles) which the SO stage has or would have written to a particular "stream's" Streamed Vertex Output buffers if all buffers had been large enough to accommodate the writes. (See the Stream Output section of the 3D pipeline volume).

These registers are part of the context save and restore.

DWord	Bit	Description
0	63:0	**Prim Storage Needed Count** Project: CHV, BSW Format: U64 This count is incremented (by one) by the SOL stage for each object (point, line, triangle) it writes or attempts to write to the corresponding stream's output buffers. The count is not affected by the actual number of buffers bound to the stream.

Stream Output Write Offsets

SO_WRITE_OFFSET[0:3] - Stream Output Write Offsets

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RW. This register is set by the context restore.
Size (in bits):	32
Address:	05280h-0528Fh

There is one R/W 32-bit register for each of the 4 supported stream output buffer slots:
5280h-5283h SO_WRITE_OFFSET0 (for Stream Out Buffer #0)
5284h-5287h SO_ WRITE_OFFSET1 (for Stream Out Buffer #1)
5288h-528Bh SO_ WRITE_OFFSET2 (for Stream Out Buffer #2)
528Ch-528Fh SO_ WRITE_OFFSET3 (for Stream Out Buffer #3)
These registers are used to set and track a DWord-granular Write Offset for each of the 4 Stream Output Buffer slots. Software can directly write them via MI_LOAD_REGxxx commands. The SOL stage will increment them as part of stream output processing. Software can cause them to be written to memory via MI_STORE_REGxxx commands. (See the Stream Output section of the 3D pipeline volume).
These registers are part of the context save and restore.

Programming Notes

- Software must ensure that no HW stream output operations can be in process or otherwise pending at the point that the MI_LOAD/STORE commands are processed. This will likely require a pipeline flush.
- The SOL stage will effectively advance the write offset by the buffer's Surface Pitch after each vertex is written (assuming no overflow is detected in any targetted SO buffer). Under "normal" conditions one would expect software to initialize the WriteOffset to some (possibly zero) multiple of Surface Pitch in order to align vertex writes to the buffer's Base Address, though it is not required to do so.

DWord	Bit	Description
0	31:2	**Write Offset** Project: CHV, BSW Format: U30 This field contains a DWord offset from the corresponding SO buffer's Base Address value. The SOL stage uses this value as a write offset when performing writes to the buffer. The SOL stage will increment this value as a part of performing stream output to the buffer. Note that the SOL stage uses the buffer's Surface Pitch to advance the Write Offset, without regard to the buffer's Base Address (see Programming Notes above).
	1:0	**Reserved** Format: MBZ

SWF1

SWF1 - SWF1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F000h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF2

SWF2 - SWF2	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F004h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF3

SWF3 - SWF3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F008h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF4

SWF4 - SWF4	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F00Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF5

SWF5 - SWF5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F010h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF6

SWF6 - SWF6	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F014h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF7

SWF7 - SWF7

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F018h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF8

SWF8 - SWF8	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F01Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF9

SWF9 - SWF9

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F020h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF10

SWF10 - SWF10	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F024h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF11

SWF11 - SWF11

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F028h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF12

SWF12 - SWF12	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F02Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF13

SWF13 - SWF13

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F030h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF14

SWF14 - SWF14	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F034h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF15

SWF15 - SWF15	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F038h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF16

SWF16 - SWF16	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F03Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF17

SWF17 - SWF17

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F040h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF18

SWF18 - SWF18	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F044h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF19

SWF19 - SWF19

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F048h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF20

SWF20 - SWF20	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F04Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF21

SWF21 - SWF21	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F050h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF22

SWF22 - SWF22	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F054h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF23

SWF23 - SWF23

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F058h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF24

SWF24 - SWF24	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F05Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF25

SWF25 - SWF25

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F060h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF26

SWF26 - SWF26	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F064h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF27

SWF27 - SWF27

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F068h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF28

SWF28 - SWF28	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F06Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF29

SWF29 - SWF29

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F070h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF30

SWF30 - SWF30	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F074h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF31

SWF31 - SWF31

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F078h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF32

SWF32 - SWF32	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F07Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF33

SWF33 - SWF33

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F080h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF34

SWF34 - SWF34	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F084h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF35

SWF35 - SWF35

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F088h

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWF36

SWF36 - SWF36	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	4F08Ch

These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation.
The use of these register is defined by software architecture.

DWord	Bit	Description
0	31:0	**SOFTWARE_FLAGS** Default Value: 00000000h Access: R/W These (36 DW) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these register is defined by software architecture.

SWSMISCI

SWSMISCI - SWSMISCI

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	000E0h

Software SMI or SCI.

The SCI mechanism for driver / BIOS communication. SMI is a system wide lock interrupt (halts all the cores) as opposed to SCI.

The SMI is slowly being phased out.
This register serves 2 purposes: 1) Support selection of SMI or SCI event source (SMISCISEL - bit15) 2) Event trigger (bit 0).
To generate a SW SCI event, software (System BIOS/Graphics driver) should program bit 15 (SMISCISEL) to 1. This is typically programmed once (assuming SMIs are never triggered). On a '0' to '1' subsequent transition in bit 0 of this register (caused by a software write operation), GMCH sends a single SCI message. The SCI will set the DMISCI bit in its TCO1_STS register and TCOSCI_STS bit in its GPE0 register upon receiving this message from DMI.
Once written as 1, software must write a '0' to this bit to clear it, and all other write transitions (1->0, 0->0, 1->1) or if bit 15 is '0' will not cause GMCH to send SCI message to DMI link.
To generate a SW SMI event, software should program bit 15:0 and trigger SMI.

DWord	Bit	Description
0	31:16	**RESERVED**
	15	**SMI_OR_SCI_EVENT_SELECT** Default Value: 0b Access: R/W MCS: SMI or SCI event select. 0 = SMI,1 = SCI
	14:1	**SOFTWARE_SCRATCH_BITS** Default Value: 0000h Access: R/W Used by driver to communicate information to SBIOS
	0	**SMI_OR_SCI_EVENT** Default Value: 0b Access: R/W MCE: MCS=1, setting this bit causes an SCI. MCS=0, setting this bit causes an SMI. A 1 to 0, 0 to 0 or 1 to 1 transition of this bit does not trigger any events. The graphics driver writes to this register as a means to interrupt the SBIOS.

Thread Dispatched Count Register

TDL_THR_DISP_COUNT - Thread Dispatched Count Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E4BCh
Valid Projects:	CHV, BSW

This register provides the count of threads dispatched/valid in the subslice.

DWord	Bit	Description
0	31:6	**Reserved** Format: MBZ
	5:0	**Thread Count** Value: 0-56 — Name: Valid Range

Thread Faulted Count Register

TDL_THR_PF_COUNT - Thread Faulted Count Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E5BCh
Valid Projects:	CHV, BSW

This register provides the count of threads faulted in each subslice.

DWord	Bit	Description
0	31	**Canonical fault indication bit to CS** The bit is set when a canonical fault on data fetch is reported by EU.
	30:6	**Reserved** Format: MBZ
	5:0	**Thread Count** Value: 0-56 — Name: Valid Range

Thread Fault Status Register 0

TDL_THR_PF_STATUS0 - Thread Fault Status Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E6B8h

This register provides the status of each thread in the SubSlice. A bit set indicates the thread in the specific thread slot is faulted.

DWord	Bit	Description
0	31:24	**Row0, EU3, [Reserved, T6-T0]**
	23:16	**Row0, EU2, [Reserved, T6-T0]**
	15:8	**Row0, EU1, [Reserved, T6-T0]**
	7:0	**Row0, EU0, [Reserved, T6-T0]**

Thread Fault Status Register 1

TDL_THR_PF_STATUS1 - Thread Fault Status Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E7B8h

This register provides the status of each thread in the SubSlice. A bit set indicates the thread in the specific thread slot is faulted.

DWord	Bit	Description
0	31:24	**Row1, EU3, [Reserved, T6-T0]**
	23:16	**Row1, EU2, [Reserved, T6-T0]**
	15:8	**Row1, EU1, [Reserved, T6-T0]**
	7:0	**Row1, EU0, [Reserved, T6-T0]**

Thread Load Status Register 0

TDL_THR_STATUS0 - Thread Load Status Register 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E4B8h
This register provides the status of each thread in the SubSlice.	

DWord	Bit	Description
0	31:24	**Row0, EU3, [Reserved, T6-T0]**
	23:16	**Row0, EU2, [Reserved, T6-T0]**
	15:8	**Row0, EU1, [Reserved, T6-T0]**
	7:0	**Row0, EU0, [Reserved, T6-T0]**

Thread Load Status Register 1

TDL_THR_STATUS1 - Thread Load Status Register 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	0E5B8h

This register provides the status of each thread in the SubSlice. A bit set indicates a valid thread is loaded in the thread slot.

DWord	Bit	Description
0	31:24	**Row1, EU3, [Reserved, T6-T0]**
	23:16	**Row1, EU2, [Reserved, T6-T0]**
	15:8	**Row1, EU1, [Reserved, T6-T0]**
	7:0	**Row1, EU0, [Reserved, T6-T0]**

Thread Mode Register

FF_MODE - Thread Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00A00000 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	020A0h
Valid Projects:	

This register is used to program the FF shader Mode.

DWord	Bit	Description
0	31	**Reserved** Project: CHV, BSW Format: MBZ
	30	**TDS external Cache Disable** Project: CHV, BSW Value / Name / Description: 0b / Enable **[Default]** / The external TDS Cache is enabled if there is enough handles to enable the cache. 1b / Disable / The external TDS Cache is disabled even if there is enough handles to enable the cache. Only the internal TDS Cache will be used.
	29:26	**DS Hit Max Value** Format: U4 Description / Project: If the number of hits reaches the DS Hit Max Value and there is a pending miss to be dispatched, the DS will dispatch the pending miss vertex as a single dispatch. / Programming the value beyond the range will have undefined behavior if DS Reference Count Full Force miss enable is 0. When DS Reference Count Full Force miss enable is 1 then the value can be [1, Fh]. / CHV, BSW Value / Name / Project: 9 / **[Default]** / CHV, BSW [1,9] / / CHV, BSW

FF_MODE - Thread Mode Register

25:20 **VS Hit Max Value**

Format:	U6

Description
If the number of hits reaches the VS Hit Max Value and there is a pending miss to be dispatched, the VS will dispatch the pending miss vertex as a single dispatch.
Programming the value beyond the range will have undefined behavior if VS Reference Count Full Force miss enable is 0. When VS Reference Count Full Force miss enable is 1 then the value can be [1,3Fh].

Value	Name
10	**[Default]**
[1,26]	

19 **DS Reference Count Full Force Miss Enable**

Project:	CHV, BSW
Format:	Enable

Value	Name	Description
0b	**[Default]**	On a hit to the DS cache and the associated handle's reference count is full then stall until a derefernce.
1b		On a hit to the DS cache and the associated handle's reference count is full then force the cycle as a miss and allocate a new handle.

Programming Notes
To work around bugs, this must be set to 0.

18 **Reserved**

17:16 **Reserved**

15 **VS Reference Count Full Force Miss Enable**

Format:	U1

Value	Name	Description
[0,1]		
0b	**[Default]**	On a hit to the VS cache and the associated handle's reference count is full then stall until a derefernce.
1b		On a hit to the VS cache and the associated handle's reference count is full then force the cycle as a miss and allocate a new handle.

Programming Notes	Project
To work around bugs, this must be set to 0.	CHV, BSW

FF_MODE - Thread Mode Register

	14:13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**Reserved** Default Value: 0h Project: CHV, BSW Format: MBZ
	11:7	**Reserved** Format: MBZ
	6:5	**Reserved** Project: CHV, BSW Format: MBZ
	4	**Reserved** Default Value: 0h Project: CHV, BSW Format: MBZ
	3:0	**Reserved** Format: MBZ

Thread Restart Control Register

TDL_THR_RESTART - Thread Restart Control Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	WO
Size (in bits):	32
Address:	0E450h

This register provides control to restart page faulted and halted threads in each subslice.

DWord	Bit	Description
0	31:1	**Reserved** Format: MBZ
	0	**Restart All Faulted Threads** A write of 1 to this register restarts all threads that have halted due to page fault.

TILECTL

TILECTL - TILECTL	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	101000h

DWord	Bit	Description
0	31:3	**Reserved** Default Value: 00000000h Access: RO Reserved
	2	**Reserved** Default Value: 0b Access: RO Reserved.
	1	**TLBPF** Default Value: 0b Access: R/W Store multiple PTE enable 0: Only one Page Table Entry is stored in the Translation Lookaside Table Entry is stored in the Translation Lookaside Buffer cache for tiled cycles. 1: Multiple Page Table Entries are stored in the Translation Lookaside Buffer cache for tiled cycles. If tileX, then 4 entries are stored. If tileY, then 8 entries are stored.
	0	**SWZCTL** Default Value: 0b Access: R/W Not used for CHV, BSW. This register location is updated via GFX Driver prior to enabling DRAM accesses. The Driver needs to obtain the need for memory address swizzling via DRAM configuration registers and set the following bits. x0b - No Address Swizzling x1b - Address bit [6] needs to be swizzled for tiled surfaces

TiledResources Invalid Tile Detection Register

TRINVTILEDETCT - TiledResources Invalid Tile Detection Register

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DECh
Name:	TiledResources Invalid Tile Detection Register
ShortName:	TRINVTILEDETCT

DWord	Bit	Description
0	31:0	**Invalid Tile Detection Value** Access: R/W **Value** / **Name** / **Description** 00000000h / **[Default]** / A 32bit value programmed to enable h/w to perform a match of TR-VA TT entries to detect Invalid Tiles. Hardware will flag each entry and space behind it as Invalid Tile for matched entries.

Tiled Resources Translation Table Control Registers

TRTTE - Tiled Resources Translation Table Control Registers

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DF4h
Name:	Tiled Resources Translation Table Control Register
ShortName:	TRTTE

DWord	Bit	Description
0	31:2	**Reserved** Access: RO Value: 00000000000000000000000000000000b — **[Default]** — Reserved
	1	**TR-VA Translation Table Memory Location** Default Value: 0b Access: R/W This field specifies whether the translation tables for TR to VA are in virtual address space v/s physical (GPA) address space. 0: Tables are in Physical (GPA) space 1: Tables are in Virtual address space
	0	**TR - TT Enable** Default Value: 0b Access: R/W TR translation tables are disabled as default. This field needs to be enabled via s/w to get TR translation active.

TiledResources VA Detection Registers

TRVADR - TiledResources VA Detection Registers

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DF0h
Name:	TiledResources VA Detection Registers
ShortName:	TRVADR

DWord	Bit	Description
0	31:8	**Reserved** Default Value: 000000h Access: RO
	7:4	**TR - VA Mask Value** Default Value: 0000b Access: R/W 4bit MASK value that is mapped to incoming address bits[47:44] MASK bits are used to identify which address bits need to be considered for compare. If particular mask bit is "1", mapping address bit needs to be compared to DATA value provided. If "0", corresponding address bit is masked which makes it don't care for compare. (This field defaults to "0000" to disable detection.). Note: The only usage model for GFX driver to set this field to "1111". Behaviour of h/w for any other setiing is not defined. Note: GFX driver shall use same TRVA MASK value for all contexts.
	3:0	**TR- VA Data Value** Access: R/W Value / Name / Description: 0000b **[Default]** 4bit Data value that is mapped to incoming address bits[47:44]. Data bits are used to compare address values that are not filtered by the TRVAMV for match Note: GFX driver shall use same TRVA Data value for all contexts

Tiled Resources VA Translation Table L3 ptr - DW0

TRVATTL3PTRDW0 - Tiled Resources VA Translation Table L3 ptr - DW0

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DE0h
Name:	Tiled Resources VA Translation Table L3 ptr - DW0
ShortName:	TRVATTL3PTRDW0

DWord	Bit	Description
0	31:12	**TR - VA transln Table L3 Pointer (Lower Address)** Access: R/W Value / Name / Description: 00000h **[Default]** Lower address bits for tiled resource VA to virtual address translation L3 table
	11:0	**Reserved** Default Value: 000h Access: RO Reserved

Tiled Resources VA Translation Table L3 ptr - DW1

TRVATTL3PTRDW1 - Tiled Resources VA Translation Table L3 ptr - DW1

Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DE4h
Name:	Tiled Resources VA Translation Table L3 ptr - DW1
ShortName:	TRVATTL3PTRDW1

DWord	Bit	Description
0	31:16	**Reserved** Access: RO Value: 0000h — Name: **[Default]** — Description: Reserved
	15:0	**TR - VA transln Table L3 Pointer (Upper Address)** Default Value: 0000h Access: R/W Upper address bits for tiled resource VA to virtual address translation L3 table

TLB_RD_ADDRESS Register

TLB_RD_ADDR - TLB_RD_ADDRESS Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B00h

DWord	Bit	Description
0	31:12	**Reserved** Default Value: 00000000000000000000b Access: RO
	11:0	**Reserved**

TLB_RD_DATA0 Register

TLB_RD_DATA0 - TLB_RD_DATA0 Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B04h

DWord	Bit	Description
0	31:0	**TLB_READ_DATA0 Register** Default Value: 00000000h Access: RO address [43:12]

TLB_RD_DATA1 Register

TLB_RD_DATA1 - TLB_RD_DATA1 Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04B08h	
DWord	**Bit**	**Description**
0	31:0	**TLB_READ_DATA1 Register** Default Value: 00000000h Access: RO Bit[31:5] Reserved Bit[4] Cycle GTT SEL (1-GGTT Cycle, 0-PPGTT Cycle) Bit[3:0] address [47:44]

Transcode Attack Status Register

TRANS_STAT - Transcode Attack Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09034h

Transcode Attack Status Register

DWord	Bit	Description
0	31:1	**RSVD** Access: RO RSVD
	0	**GKEYS_STATUS** Access: RO Transcode Attack Status bit

TRNULLDETCT

REG_TEMPLATE - TRNULLDETCT	
Register Space:	MMIO: 0/2/0
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DE8h
Name:	TiledResources Null Tile Detection Register
ShortName:	TRNULLDETCT

DWord	Bit	Description
0	31:0	**Null Tile Detection Value** Access: R/W **Value** / **Name** / **Description** 00000000h / **[Default]** / A 32bit value programmed to enable h/w to perform a match of TR-VA TT entries to detect Null Tiles. Hardware will flag each entry and space behind it as Null Tile for matched entries.

Turbo Media Control

TMCTL - Turbo Media Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A190h-0A193h

DWord	Bit	Description
0	31:9	**Reserved** Access: RO
	8	**Bypass Media Idle Hysteresis Enable (aka Slice Shutdown)** Access: R/W 1 - Bypass idle hysteresis requirements for making an RC wish. 0 - Honor idle hysteresis <default>
	7:1	**Reserved** Access: RO
	0	**Bypass Render Idle Hysteresis Enable (aka Slice Shutdown)** Access: R/W 1 - Bypass idle hysteresis requirements for making an RC wish. 0 - Honor idle hysteresis <default>

Unblock Message Act to Busy Detection Timer

RCUBMABDTMR - Unblock Message Act to Busy Detection Timer

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0A0B0h-0A0B3h

When locked exit policy is chosen, the GT core can be woken up in parallel with the IA cores waking up so an MMIO write to GT core might not be coming, or may be later in coming. If after the time programmed in this register is met and graphics is still idle, then RC1(e)/RC6x is allowed to be entered.
0 = 0 usec
1 = 1.28 usec
2 = 2.56 usec
3 = 3.84 usec
FF FFFF = 21.474 sec
pmcr_rc_sleep[23:0]

DWord	Bit	Description
0	31:24	**Reserved** Access: RO Reserved
	23:0	**Unblock Message Ack to Busy Detection Timer** Access: R/W When locked exit policy is chosen, the GT core can be woken up in parallel with the IA cores waking up so an MMIO write to GT core might not be coming, or may be later in coming. If after the time programmed in this register is met and graphics is still idle, then RC1(e)/RC6x is allowed to be entered. 0 = 0 usec 1 = 1.28 usec 2 = 2.56 usec 3 = 3.84 usec FF FFFF = 21.474 sec pmcr_rc_sleep[23:0]

Unit Level Clock Gating Control 1

UCGCTL1 - Unit Level Clock Gating Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x02800000
Size (in bits):	32
Address:	09400h

Unit Level Clock Gating Control Registers.Refer to the Programming notes mentioned near GT Interface Registers in PRM

DWord	Bit	Description
0	31	**Sarbunit Clock Gating Disable** Access: R/W SARB unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**Reserved** Access: R/W
	29	**Reserved** Access: R/W
	28	**ICunit Clock Gating Disable** Access: R/W ICunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**HIZunit Clock Gating Disable** Access: R/W HIZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**GWunit Clock Gating Disable** Access: R/W GWunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

25	**GTIunit Clock Gating Disable** Default Value: 1b Access: R/W GTI Units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
24	**GSunit Clock Gating Disable** Access: R/W GSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
23	**GPMunit Clock Gating Disable** Default Value: 1b Access: R/W GPMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
22	**GAMunit Clock Gating Disable** Default Value: 0b Access: R/W GAMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
21	**GACunit Clock Gating Disable** Default Value: 0b Access: R/W GACunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

20	**GABunit Clock Gating Disable** Default Value: 0b Access: R/W GABunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
19	**FTunit Clock Gating Disable** Access: R/W FTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
18	**FLunit Clock Gating Disable** Access: R/W FLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
17	**EU_FPUunit Clock Gating Disable** Access: R/W EU_FPUunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
16	**EU_TCunit Clock Gating Disable** Access: R/W EU_TCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
15	**EU_EMunit Clock Gating Disable** Access: R/W EU_EMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

14	**EU_GAunit Clock Gating Disable** Access: R/W EU_GAunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
13	**EUunit Clock Gating Disable** Access: R/W EUunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
12	**SVLunit Clock Gating Disable** Access: R/W SVLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
11	**DTunit Clock Gating Disable** Access: R/W DTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
10	**DMunit Clock Gating Disable** Access: R/W DMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
9	**DGunit Clock Gating Disable** Access: R/W DGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

	8	**DAPunit Clock Gating Disable** Access: R/W DAPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**CSunit Clock Gating Disable** Access: R/W CSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**CLunit Clock Gating Disable** Access: R/W CLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**BLBunit Clock Gating Disable** Access: R/W BLBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**BFunit Clock Gating Disable** Access: R/W BFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**BDunit Clock Gating Disable** Access: R/W BDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

	2	**BCSunit Clock Gating Disable** Access: R/W BCSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**AVSunit Clock Gating Disable** Access: R/W AVSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**SPARE RAM Clock Gating Disable** Access: R/W SPARE RAM Clock Gating Disable Control: '0' : RAM Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : RAM Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 1

UCGCTL1 - Unit Level Clock Gating Control 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x02F00000
Size (in bits):	32
Address:	09400h

Unit Level Clock Gating Control Registers.

DWord	Bit	Description
0	31	**Sarbunit Clock Gating Disable** Access: R/W SARB unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**Reserved** Access: R/W
	29	**Reserved** Access: R/W
	28	**ICunit Clock Gating Disable** Access: R/W ICunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**HIZunit Clock Gating Disable** Access: R/W HIZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**GWunit Clock Gating Disable** Access: R/W GWunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

25	**GTIunit Clock Gating Disable** Default Value: 1b Access: R/W GTI Units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
24	**GSunit Clock Gating Disable** Access: R/W GSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
23	**GPMunit Clock Gating Disable** Default Value: 1b Access: R/W GPMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
22	**GAMunit Clock Gating Disable** Default Value: 1b Access: R/W GAMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
21	**GACunit Clock Gating Disable** Default Value: 1b Access: R/W GACunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
20	**GABunit Clock Gating Disable** Default Value: 1b

UCGCTL1 - Unit Level Clock Gating Control 1

		Access: R/W GABunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**FTunit Clock Gating Disable** Access: R/W FTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**FLunit Clock Gating Disable** Access: R/W FLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**EU_FPUunit Clock Gating Disable** Access: R/W EU_FPUunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**EU_TCunit Clock Gating Disable** Access: R/W EU_TCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**EU_EMunit Clock Gating Disable** Access: R/W EU_EMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

	14	**EU_GAunit Clock Gating Disable** Access: R/W EU_GAunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**EUunit Clock Gating Disable** Access: R/W EUunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**SVLunit Clock Gating Disable** Access: R/W SVLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**DTunit Clock Gating Disable** Access: R/W DTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**DMunit Clock Gating Disable** Access: R/W DMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**DGunit Clock Gating Disable** Access: R/W DGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

	8	**DAPunit Clock Gating Disable** Access: R/W DAPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**CSunit Clock Gating Disable** Access: R/W CSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**CLunit Clock Gating Disable** Access: R/W CLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**BLBunit Clock Gating Disable** Access: R/W BLBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**BFunit Clock Gating Disable** Access: R/W BFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**BDunit Clock Gating Disable** Access: R/W BDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL1 - Unit Level Clock Gating Control 1

	2	**BCSunit Clock Gating Disable** Access: R/W BCSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**AVSunit Clock Gating Disable** Access: R/W AVSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**SPARE RAM Clock Gating Disable** Access: R/W SPARE RAM Clock Gating Disable Control: '0' : RAM Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : RAM Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 2

UCGCTL2 - Unit Level Clock Gating Control 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09404h

Unit Level Clock Gating Control Registers.Refer to the Programming notes mentioned near GT Interface Registers in PRM

DWord	Bit	Description
0	31	**VFunit Clock Gating Disable** Access: R/W VFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**VDSunit Clock Gating Disable** Access: R/W VDSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VDIunit Clock Gating Disable** Access: R/W VDIunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**VCSunit Clock Gating Disable** Access: R/W VCSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	27	**DTOunit Clock Gating Disable** Access: R/W DTOunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**VCPunit Clock Gating Disable** Access: R/W VCPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**VCDunit Clock Gating Disable** Access: R/W VCDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**URBMunit Clock Gating Disable** Access: R/W URBMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**TSGunit Clock Gating Disable** Access: R/W TSGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**TDLunit Clock Gating Disable** Access: R/W TDLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	21	**TDSunit Clock Gating Disable** Access: R/W TDSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**SVSMunit Clock Gating Disable** Access: R/W SVSMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**SVGunit Clock Gating Disable** Access: R/W SVGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**SOunit Clock Gating Disable** Access: R/W SOunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**SIunit Clock Gating Disable** Access: R/W SIunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**SFunit Clock Gating Disable** Access: R/W SFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

15	**SECunit Clock Gating Disable** Access: R/W SECunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
14	**SCunit Clock Gating Disable** Access: R/W SCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
13	**RCZunit Clock Gating Disable** Access: R/W RCZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
12	**RCPBunit Clock Gating Disable** Access: R/W RCPBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
11	**RCCunit Clock Gating Disable** Access: R/W RCCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
10	**QCunit Clock Gating Disable** Access: R/W QCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	9	**PSDunit Clock Gating Disable** Access: R/W PSDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**PLunit Clock Gating Disable** Access: R/W PLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**MTunit Clock Gating Disable** Access: R/W MTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**MPCunit Clock Gating Disable** Access: R/W MPCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**TDGunitClock Gating Disable** Access: R/W TDGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**MSCunit Clock Gating Disable** Access: R/W MSCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	3	**TEunit Clock Gating Disable** Access: R/W TEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**TETGunit Clock Gating Disable** Access: R/W TETGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**MAunit Clock Gating Disable** Access: R/W MAunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**IZunit Clock Gating Disable** Access: R/W IZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 2

UCGCTL2 - Unit Level Clock Gating Control 2

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09404h

Unit Level Clock Gating Control Registers.

DWord	Bit	Description
0	31	**VFunit Clock Gating Disable** Access: R/W VFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**VDSunit Clock Gating Disable** Access: R/W VDSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VDIunit Clock Gating Disable** Access: R/W VDIunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**VCSunit Clock Gating Disable** Access: R/W VCSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	27	**DTOunit Clock Gating Disable** Access: R/W DTOunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**VCPunit Clock Gating Disable** Access: R/W VCPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**VCDunit Clock Gating Disable** Access: R/W VCDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**URBMunit Clock Gating Disable** Access: R/W URBMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**TSGunit Clock Gating Disable** Access: R/W TSGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**TDLunit Clock Gating Disable** Access: R/W TDLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	21	**TDSunit Clock Gating Disable** Access: R/W TDSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**SVSMunit Clock Gating Disable** Access: R/W SVSMunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**SVGunit Clock Gating Disable** Access: R/W SVGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**SOunit Clock Gating Disable** Access: R/W SOunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**SIunit Clock Gating Disable** Access: R/W SIunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**SFunit Clock Gating Disable** Access: R/W SFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

15	**SECunit Clock Gating Disable** Access: R/W SECunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
14	**SCunit Clock Gating Disable** Access: R/W SCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
13	**RCZunit Clock Gating Disable** Access: R/W RCZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
12	**RCPBunit Clock Gating Disable** Access: R/W RCPBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
11	**RCCunit Clock Gating Disable** Access: R/W RCCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
10	**QCunit Clock Gating Disable** Access: R/W QCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	9	**PSDunit Clock Gating Disable** Access: R/W PSDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**PLunit Clock Gating Disable** Access: R/W PLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**MTunit Clock Gating Disable** Access: R/W MTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**MPCunit Clock Gating Disable** Access: R/W MPCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**TDGunitClock Gating Disable** Access: R/W TDGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**MSCunit Clock Gating Disable** Access: R/W MSCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL2 - Unit Level Clock Gating Control 2

	3	**TEunit Clock Gating Disable** Access: R/W TEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**TETGunit Clock Gating Disable** Access: R/W TETGunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**MAunit Clock Gating Disable** Access: R/W MAunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**IZunit Clock Gating Disable** Access: R/W IZunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 3

UCGCTL3 - Unit Level Clock Gating Control 3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000 [CHV:B, CHV:C, CHV:K]
	0x04000000 [CHV:A]
Size (in bits):	32
Address:	09408h

Unit Level Clock Gating Control Registers.Refer to the Programming notes mentioned near GT Interface Registers in PRM

DWord	Bit	Description
0	31	**Flunits 2nd Clock Gating Disable** Access: R/W Flunits 2nd Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**SVRRunit Clock Gating Disable** Access: R/W SVRRunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VCRunit Clock Gating Disable** Access: R/W VCRunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**EDTunit Clock Gating Disable** Access: R/W EDTunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

27	**VCIunit Clock Gating Disable** Access: R/W VCIunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
26	**HEVC DOP Gating Enable** Default Value: 1b Project: CHV, BSW Access: R/W HEVCUNIT' Clock Gating Enable Control: '0' : VIDPAR8/VIDPAR9 DOP gating is not affected by this bit (i.e., this bit does not affect functional DOP gating) '1' : VIDPAR8/VIDPAR9 cmclk/cuclk DOP's are gated. (i.e., functional clocks aren't toggling, always)
26	**HEVC DOP Gating Enable** Default Value: 0b Project: CHV, BSW Access: R/W HEVCUNIT' Clock Gating Enable Control: '0' : VIDPAR8/VIDPAR9 DOP gating is not affected by this bit (i.e., this bit does not affect functional DOP gating) '1' : VIDPAR8/VIDPAR9 cmclk/cuclk DOP's are gated. (i.e., functional clocks aren't toggling, always)
25	**HSunit Clock Gating Disable** Access: R/W HSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
24	**SOLunit Clock Gating Disable** Access: R/W SOLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	23	**QRCunit Clock Gating Disable** Access: R/W QRCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**MSPBISTunit Clock Gating Disable** Access: R/W MSPBISTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	21	**BSPunit Clock Gating Disable** Access: R/W BSPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**Reserved**
	19	**SBEunit Clock Gating Disable** Access: R/W SBEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**BCunit Clock Gating Disable** Access: R/W BCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**WMBE Clock Gating Disable** Access: R/W WMBEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	16	**WMFEunit Clock Gating Disable** Access: R/W WMFEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**VSCunit Clock Gating Disable** Access: R/W VSCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**Reserved**
	13	**USBunit Clock Gating Disable** Access: R/W USBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**STCunit Clock Gating Disable** Access: R/W STCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**VSunit Clock Gating Disable** Access: R/W VSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VOPunit Clock Gating Disable** Access: R/W VOPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	9	**VMXunit Clock Gating Disable** Access: R/W VMXunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**VMEunit Clock Gating Disable** Access: R/W VMEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**VMDunit Clock Gating Disable** Access: R/W VMDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VMCunit Clock Gating Disable** Access: R/W VMCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VLFunit Clock Gating Disable** Access: R/W VLFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**VITunit Clock Gating Disable** Access: R/W VITunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	3	**VIPunit Clock Gating Disable** Access: R/W VIPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**VINunit Clock Gating Disable** Access: R/W VINunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**VFTunit Clock Gating Disable** Access: R/W VFTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**VFEunit Clock Gating Disable** Access: R/W VFEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 3

UCGCTL3 - Unit Level Clock Gating Control 3

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x04000000
Size (in bits):	32
Address:	09408h

Unit Level Clock Gating Control Registers.

DWord	Bit	Description
0	31	**Flunits 2nd Clock Gating Disable** Access: R/W Flunits 2nd Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**SVRRunit Clock Gating Disable** Access: R/W SVRRunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VCRunit Clock Gating Disable** Access: R/W VCRunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**EDTunit Clock Gating Disable** Access: R/W EDTunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	27	**VCIunit Clock Gating Disable** Access: R/W VCIunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**2x Assign fub XOR Clock Gating Disable** Default Value: 1b Access: R/W 2x Assign fub XOR Clock Gating Disable Control: '0' : 2x Assign fub XOR Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : 2x Assign fub XOR Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**HSunit Clock Gating Disable** Access: R/W HSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**SOLunit Clock Gating Disable** Access: R/W SOLunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**QRCunit Clock Gating Disable** Access: R/W QRCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**MSPBISTunit Clock Gating Disable** Access: R/W MSPBISTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	21	**BSPunit Clock Gating Disable** Access: R/W BSPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**OACSunit Clock Gating Disable** Access: R/W OACSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**SBEunit Clock Gating Disable** Access: R/W SBEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**BCunit Clock Gating Disable** Access: R/W BCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**WMBE Clock Gating Disable** Access: R/W WMBEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**WMFEunit Clock Gating Disable** Access: R/W WMFEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	15	**VSCunit Clock Gating Disable** Access: R/W VSCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**Reserved** Access: R/W
	13	**USBunit Clock Gating Disable** Access: R/W USBunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**STCunit Clock Gating Disable** Access: R/W STCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**VSunit Clock Gating Disable** Access: R/W VSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VOPunit Clock Gating Disable** Access: R/W VOPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	9	**VMXunit Clock Gating Disable** Access: R/W VMXunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**VMEunit Clock Gating Disable** Access: R/W VMEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**VMDunit Clock Gating Disable** Access: R/W VMDunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VMCunit Clock Gating Disable** Access: R/W VMCunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VLFunit Clock Gating Disable** Access: R/W VLFunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**VITunit Clock Gating Disable** Access: R/W VITunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL3 - Unit Level Clock Gating Control 3

	3	**VIPunit Clock Gating Disable** Access: R/W VIPunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**VINunit Clock Gating Disable** Access: R/W VINunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**VFTunit Clock Gating Disable** Access: R/W VFTunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**VFEunit Clock Gating Disable** Access: R/W VFEunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 4

UCGCTL4 - Unit Level Clock Gating Control 4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00F80003
Size (in bits):	32
Address:	0940Ch

Unit Level Clock Gating Control Registers.Refer to the Programming notes mentioned near GT Interface Registers in PRM

DWord	Bit	Description
0	31:30	**Reserved** Access: RO Reserved.
	29	**GAFSRRB unit Clock Gate Disable** Access: R/W GAFSRRB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**RAMDFT units Clock Gate Disable** Access: R/W RAMDFT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**TDCunit Clock Gate Disable** Access: R/W TDCunit Clock Gating Disable Control : '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**L3 CBR 1x Clock Gate Disable** Access: R/W L3 CBR units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

25	**L3 BANK 2x Clock Gate Disable** Access: R/W L3 BANK units 2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
24	**L3 BANK 1x Clock Gate Diable** Access: R/W L3 BANK units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
23	**MBGFunit Clock Gate Disable** Default Value: 1b Access: R/W MBGFunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
22	**MSQDunit 2x Clock Gate Disable** Default Value: 1b Access: R/W MSQD units cu2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
21	**MSQDunit Clock Gate Disable** Default Value: 1b Access: R/W MSQD units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
20	**MISDunits 2x Clock Gate Disable** Default Value: 1b Access: R/W MISDunits cu2x Clock Gating Disable Control:

UCGCTL4 - Unit Level Clock Gating Control 4

		'0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**MISDunit Clock Gate Disable** Default Value: 1b Access: R/W MISDunits 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**GAFMunit Clock Gate Disable** Access: R/W GAFMunit' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**GAPCunit Clock Gate Disable** Access: R/W GAPCunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**GAPZunit Clock Gate Disable** Access: R/W GAPZunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**GAPL3unit Clock Gate Disable** Access: R/W GAPL3 units' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	14	**GAFSunit Clock Gate Disable** Access: R/W GAFSunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**GAHSunit Clock Gate Disable** Access: R/W GAHSunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**VISunit Clock Gate Disable** Access: R/W VISunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**VACunit Clock Gate Disable** Access: R/W VACunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VAMunit Clock Gate Disable** Access: R/W VAMunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VADuit Clock Gating Disable** Access: R/W VADunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	8	**JPGunit Clock Gating Disable** Access: R/W JPGunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**VBPunits Clock Gating Disable** Access: R/W VBPunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VHRunit Clock Gating Disable** Access: R/W VHRunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VID4 VINunit Clock Gating Disable** Access: R/W VID4 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**VID3 VINunit Clock Gating Disable** Access: R/W VID3 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**VID2 VINunit Clock Gating Disable** Access: R/W VID2 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	2	**VID1 VINunit Clock Gating Disable** Access: R/W VID1 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1:0	**MSQCunit Clock Gating Disable** Default Value: 11b Access: R/W MSQCunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 4

UCGCTL4 - Unit Level Clock Gating Control 4

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00F80003
Size (in bits):	32
Address:	0940Ch

Unit Level Clock Gating Control Registers.

DWord	Bit	Description
0	31:30	**Reserved** Access: RO rsvd
	29	**GAFSRRB unit Clock Gate Disable** Access: R/W GAFSRRB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**RAMDFT units Clock Gate Disable** Access: R/W RAMDFT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**L3 CBR 2x Clock Gate Disable** Access: R/W L3 CBR units 2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**L3 CBR 1x Clock Gate Disable** Access: R/W L3 CBR units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

25	**L3 BANK 2x Clock Gate Disable** Access: R/W L3 BANK units 2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
24	**L3 BANK 1x Clock Gate Diable** Access: R/W L3 BANK units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
23	**MBGFunit Clock Gate Disable** Default Value: 1b Access: R/W MBGFunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
22	**MSQDunit 2x Clock Gate Disable** Default Value: 1b Access: R/W MSQD units cu2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
21	**MSQDunit Clock Gate Disable** Default Value: 1b Access: R/W MSQD units 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
20	**MISDunits 2x Clock Gate Disable** Default Value: 1b Access: R/W MISDunits cu2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

19	**MISDunit Clock Gate Disable** Default Value: 1b Access: R/W MISDunits 1x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
18	**GAFMunit Clock Gate Disable** Access: R/W GAFMunit' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
17	**GAPCunit Clock Gate Disable** Access: R/W GAPCunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
16	**GAPZunit Clock Gate Disable** Access: R/W GAPZunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
15	**GAPL3unit Clock Gate Disable** Access: R/W GAPL3 units' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
14	**GAFSunit Clock Gate Disable** Access: R/W GAFSunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	13	**GAHSunit Clock Gate Disable** Access: R/W GAHSunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**VISunit Clock Gate Disable** Access: R/W VISunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**VACunit Clock Gate Disable** Access: R/W VACunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VAMunit Clock Gate Disable** Access: R/W VAMunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VADuit Clock Gating Disable** Access: R/W VADunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**JPGunit Clock Gating Disable** Access: R/W JPGunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	7	**VBPunits Clock Gating Disable** Access: R/W VBPunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VHRunit Clock Gating Disable** Access: R/W VHRunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VID4 VINunit Clock Gating Disable** Access: R/W VID4 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**VID3 VINunit Clock Gating Disable** Access: R/W VID3 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**VID2 VINunit Clock Gating Disable** Access: R/W VID2 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**VID1 VINunit Clock Gating Disable** Access: R/W VID1 VINunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL4 - Unit Level Clock Gating Control 4

	1:0	**MSQCunit Clock Gating Disable** Default Value: 11b Access: R/W MSQCunits' Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 5

UCGCTL5 - Unit Level Clock Gating Control 5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09418h

DWord	Bit	Description
0	31	**VCOPunit clock gating disable bit** Access: R/W WVCOP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always).
	30	**VMBunit clock gate disable bit** Access: R/W VMB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VDMunit clock gate disable bit** Access: R/W VDM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**L3BANK unit cuclkgating disable bit** Access: R/W L3bank units cuclk Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**L3BANK cu2x clock gate disable bit** Access: R/W L3BANK units cu2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks

UCGCTL5 - Unit Level Clock Gating Control 5

	26	**LNIunit clock gate disable bit** Access: R/W LNI units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**LNEUNIT clock gate disable bit** Access: R/W LNE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**VVPunit clock gate disable bit** Access: R/W VVP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**WVFT unit clock gate disable bits** Access: R/W WVFT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**WBPS unit clock gate disable bit** Access: R/W WBPS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	21	**WVMX unit clock gate disable bit** Access: R/W WVMX units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	20	**WVIP unit clock gate disable bit** Access: R/W WVIP unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**WVIT unit clock gate disable bit** Access: R/W WVIT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**WVIS unit clock gate disable** Access: R/W WVIS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**RPM units clock gate disable** Access: R/W RPM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**Reserved**
	15	**VECS unit clock gate disable** Access: R/W VECS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**GAHSV unit clock gate disable** Access: R/W GAHSV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	13	**GAHSD unit clock gate disable** Access: R/W GAHSD units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**GAV unit's clock gate disable** Access: R/W GAV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**RSunit's clock gate disable** Access: R/W RW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VFW units clock gate disable** Access: R/W VFW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VCW unit's clock gate disable** Access: R/W VCW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**VEO unit's clock gate disable** Access: R/W VEO units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	7	**VDN unit's clock gate disable** Access: R/W VDN units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VTQunit's clock gate disable** Access: R/W VTQunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VPRunit's clock gate disable** Access: R/W VPR units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**IMEunit's clock gate disable** Access: R/W IME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**CREunit clock gate disable** Access: R/W CRE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**GAPSL unit clock gate disable** Access: R/W GAPSL units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	1	**GAPSU Clock gate disable** Access: R/W GAPSU units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**SPMunit Clock gate disable** Access: R/W SPM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 5

UCGCTL5 - Unit Level Clock Gating Control 5

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09418h

DWord	Bit	Description
0	31	**VCOPunit clock gating disable bit** Access: R/W WVCOP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always).
	30	**VMBunit clock gate disable bit** Access: R/W VMB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**VDMunit clock gate disable bit** Access: R/W VDM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**L3BANK unit cuclkgating disable bit** Access: R/W L3bank units cuclk Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**L3BANK cu2x clock gate disable bit** Access: R/W L3BANK units cu2x Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks

UCGCTL5 - Unit Level Clock Gating Control 5

	26	**LNIunit clock gate disable bit** Access: R/W LNI units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**LNEUNIT clock gate disable bit** Access: R/W LNE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**VVPunit clock gate disable bit** Access: R/W VVP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**WVFT unit clock gate disable bits** Access: R/W WVFT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**WBPS unit clock gate disable bit** Access: R/W WBPS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	21	**WVMX unit clock gate disable bit** Access: R/W WVMX units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	20	**WVIP unit clock gate disable bit** Access: R/W WVIP unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**WVIT unit clock gate disable bit** Access: R/W WVIT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**WVIS unit clock gate disable** Access: R/W WVIS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**RPM units clock gate disable** Access: R/W RPM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**OASC unit clock gating disable** Access: R/W OASC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**VECS unit clock gate disable** Access: R/W VECS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	14	**GAHSV unit clock gate disable** Access: R/W GAHSV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**GAHSD unit clock gate disable** Access: R/W GAHSD units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**GAV unit's clock gate disable** Access: R/W GAV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**RSunit's clock gate disable** Access: R/W RW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**VFW units clock gate disable** Access: R/W VFW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**VCW unit's clock gate disable** Access: R/W VCW units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	8	**VEO unit's clock gate disable** Access: R/W VEO units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**VDN unit's clock gate disable** Access: R/W VDN units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**VTQunit's clock gate disable** Access: R/W VTQunits Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**VPRunit's clock gate disable** Access: R/W VPR units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**IMEunit's clock gate disable** Access: R/W IME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**CREunit clock gate disable** Access: R/W CRE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL5 - Unit Level Clock Gating Control 5

	2	**GAPSL unit clock gate disable** Access: R/W GAPSL units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**GAPSU Clock gate disable** Access: R/W GAPSU units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**SPMunit Clock gate disable** Access: R/W SPM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 6

UCGCTL6 - Unit Level Clock Gating Control 6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09430h

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	30:28	**HDCunit clock gate disable** Access: R/W HDC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**Reserved**
	26	**GACVunit cuclk gate disable** Access: R/W GACV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**GACBunit clock gate disable** Access: R/W GACB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**GAPSunit clock gate disable** Access: R/W GAPS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	23	**GAMTunit clock gate disable** Access: R/W GAMT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**Reserved**
	21	**Reserved**
	20	**Reserved**
	19	**GACVunit clock gate disable** Access: R/W GACV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**BDMunit clock gate disable** Access: R/W BDM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**GATSunit clock gate disable** Access: R/W GATS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**Reserved**
	15	**STunit clock gate disable** Access: R/W ST units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	14	**SDEunit clock gate disable** Access: R/W DE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always) **Programming Notes** Due to SDEUNIT bug HSD#1802092, this bit should be programmed to a 1 for CHV, BSW
	13	**VIN(VID6) unit clock gate disable** Access: R/W VIN(VID6) units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**VIN(VID5) unit clock gate disable** Access: R/W VIN(VID5) units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**WVOPunit clock gate disable** Access: R/W WVOP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**WUSB unit clock gate disable** Access: R/W WUSB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**WSECunit clock gate disable** Access: R/W WSEC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	8	**WRSunit clcok gate disable** Access: R/W WRS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**WQRCunit clock gate disable** Access: R/W WQRC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**WMPC unit level clock gate disable** Access: R/W WMPC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**WINunit Clock gate disable** Access: R/W WIN units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**WIME unit clock gate disable** Access: R/W WIME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**WHME unit clock gate disable** Access: R/W WHME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	2	**WAVMunit Clock Gate Disable** Access: R/W WAVM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**VSHMunit clock gate disable** Access: R/W VSHM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**VSLunit Clock gating disable** Access: R/W VSL units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 6

UCGCTL6 - Unit Level Clock Gating Control 6

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09430h

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	31	**SPARE 3 clock gate disable** Access: R/W SPARE 3 unit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30:28	**HDCunit clock gate disable** Access: R/W HDC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	27	**MUCunit clock gate disable** Access: R/W MUC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**GACVunit cuclk gate disable** Access: R/W GACV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	25	**GACBunit clock gate disable** Access: R/W GACB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**GAPSunit clock gate disable** Access: R/W GAPS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**GAMTunit clock gate disable** Access: R/W GAMT units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**Reserved**
	21	**OASCREP** Access: R/W OASCREP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**OAADDRunit clock gate disable bit** Access: R/W OAADDR units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**GACVunit clock gate disable** Access: R/W GACV units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

18	**BDMunit clock gate disable** Access: R/W BDM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
17	**GATSunit clock gate disable** Access: R/W GATS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
16	**OATREPunit clock gate disable** Access: R/W OATREP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
15	**STunit clock gate disable** Access: R/W ST units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
14	**DEunit clock gate disable** Access: R/W DE units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
13	**VIN(VID6) unit clock gate disable** Access: R/W VIN(VID6) units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	12	**VIN(VID5) unit clock gate disable** Access: R/W VIN(VID5) units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**WVOPunit clock gate disable** Access: R/W WVOP units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**WUSB unit clock gate disable** Access: R/W WUSB units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**WSECunit clock gate disable** Access: R/W WSEC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**WRSunit clcok gate disable** Access: R/W WRS units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**WQRCunit clock gate disable** Access: R/W WQRC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	6	**WMPC unit level clock gate disable** Access: R/W WMPC units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**WINunit Clock gate disable** Access: R/W WIN units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**WIME unit clock gate disable** Access: R/W WIME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**WHME unit clock gate disable** Access: R/W WHME units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**WAVMunit Clock Gate Disable** Access: R/W WAVM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**VSHMunit clock gate disable** Access: R/W VSHM units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL6 - Unit Level Clock Gating Control 6

	0	**VSLunit Clock gating disable** Access: R/W VSL units Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 7

UCGCTL7 - Unit Level Clock Gating Control 7

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09438h

DWord	Bit	Description
0	31:18	**RSVD**
	17	**Reserved**
	16	**cp_cg3ddis_hwm** Access: R/W HWM unit Clock Gating Disable (cp_cg3ddis_hwm) HWM unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	15	**cp_cg3ddis_hed** Access: R/W HED unit Clock Gating Disable (cp_cg3ddis_hed) HED unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**cp_cg3ddis_hpp** Access: R/W HPP unit Clock Gating Disable (cp_cg3ddis_hpp) HPP unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**cp_cg3ddis_hpr** Access: R/W HPR unit Clock Gating Disable (cp_cg3ddis_hpr) HPR unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality)

UCGCTL7 - Unit Level Clock Gating Control 7

		'1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**cp_cg3ddis_hmc** Access: R/W HMC unit Clock Gating Disable (cp_cg3ddis_hmc) HMC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**cp_cg3ddis_hlf** Access: R/W HLF unit Clock Gating Disable (cp_cg3ddis_hlf) HLF unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**cp_cg3ddis_hmx** Access: R/W HMX unit Clock Gating Disable (cp_cg3ddis_hmx) HMX unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	9	**cp_cg3ddis_vmm** Access: R/W VMM unit Clock Gating Disable (cp_cg3ddis_vmm) VMM unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**cp_cg3ddis_mpd** Access: R/W MPD unit Clock Gating Disable (cp_cg3ddis_mpd) MPD unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**cp_cg3ddis_mbd** Access: R/W MBD unit Clock Gating Disable (cp_cg3ddis_mbd) MBD unit Clock Gating Disable Control

UCGCTL7 - Unit Level Clock Gating Control 7

		'0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**cp_cg3ddis_mlf** Access: R/W MLF unit Clock Gating Disable (cp_cg3ddis_mlf) MLF unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**cp_cg3ddis_mmc** Access: R/W MMC unit Clock Gating Disable (cp_cg3ddis_mmc) MMC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**cp_cg3ddis_mmx** Access: R/W MMX unit Clock Gating Disable (cp_cg3ddis_mmx) MMX unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**cp_cg3ddisvbsp** Access: R/W VBSP unit Clock Gating Disable (cp_cg3ddisvbsp) VBSP unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**cp_cg3ddisvmpc** Access: R/W VMPC unit Clock Gating Disable (cp_cg3ddisvmpc) VMPC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**cp_cg3ddisvsec** Access: R/W VSEC unit Clock Gating Disable (cp_cg3ddisvsec)

UCGCTL7 - Unit Level Clock Gating Control 7

		VSEC unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**cp_cg3ddisjusb** Access: R/W JUSB unit Clock Gating Disable (cp_cg3ddisjusb) JUSB unit Clock Gating Disable Control '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 7

UCGCTL7 - Unit Level Clock Gating Control 7

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09434h

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	31	**wrcunit Clock Gating Disable** Access: R/W wrcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**mmcdunit Clock Gating Disable** Access: R/W mmcdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**bfceunit Clock Gating Disable** Access: R/W bfceunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**ecpunit Clock Gating Disable** Access: R/W ecpunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL7 - Unit Level Clock Gating Control 7

	27	**vdlunit1 Clock Gating Disable** Access: R/W vdlunit1 Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**vhmeunit Clock Gating Disable** Access: R/W vhmeunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**vimeunit Clock Gating Disable** Access: R/W vimeunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**vcreunit Clock Gating Disable** Access: R/W vcreunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**vdxunit Clock Gating Disable** Access: R/W vdxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**mdcunit Clock Gating Disable** Access: R/W mdcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL7 - Unit Level Clock Gating Control 7

	21	**hpounit Clock Gating Disable** Access: R/W hpounit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**hrsunit Clock Gating Disable** Access: R/W hrsunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**ftunit Clock Gating Disable** Access: R/W ftunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**fqunit Clock Gating Disable** Access: R/W fqunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**hleunit Clock Gating Disable** Access: R/W hleunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**hlcunit Clock Gating Disable** Access: R/W hlcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL7 - Unit Level Clock Gating Control 7

	15	**hhiunit Clock Gating Disable** Access: R/W hhiunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	14	**mlfunit Clock Gating Disable** Access: R/W mlfunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	13	**mmcunit Clock Gating Disable** Access: R/W mmcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	12	**mbdunit Clock Gating Disable** Access: R/W mbdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	11	**mpdunit Clock Gating Disable** Access: R/W mpdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	10	**mmxunit Clock Gating Disable** Access: R/W mmxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL7 - Unit Level Clock Gating Control 7

	9	**hedunit Clock Gating Disable** Access: R/W hedunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**hlfunit Clock Gating Disable** Access: R/W hlfunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**hmcunit Clock Gating Disable** Access: R/W hmcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**hmxunit Clock Gating Disable** Access: R/W hmxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	5	**hppunit Clock Gating Disable** Access: R/W hppunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**hprunit Clock Gating Disable** Access: R/W hprunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL7 - Unit Level Clock Gating Control 7

	3	**Reserved**
	2	**hwmunit Clock Gating Disable** Access: R/W hwmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**vmpcunit Clock Gating Disable** Access: R/W vmpcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**vsecunit Clock Gating Disable** Access: R/W vsecunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 8

UCGCTL8 - Unit Level Clock Gating Control 8

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	09438h

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	31	**jusbunit Clock Gating Disable** Access: R/W jusbunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	30	**sfiunit Clock Gating Disable** Access: R/W sfiunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	29	**sfeunit Clock Gating Disable** Access: R/W sfeunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	28	**sfaunit Clock Gating Disable** Access: R/W sfaunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL8 - Unit Level Clock Gating Control 8

	27	**sfounit Clock Gating Disable** Access: R/W sfounit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	26	**sfxunit Clock Gating Disable** Access: R/W sfxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	25	**sfmunit Clock Gating Disable** Access: R/W sfmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	24	**vmmunit Clock Gating Disable** Access: R/W vmmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	23	**vrtunit Clock Gating Disable** Access: R/W vrtunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	22	**ccunit Clock Gating Disable** Access: R/W ccunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL8 - Unit Level Clock Gating Control 8

	21	**gassunit Clock Gating Disable** Access: R/W gassunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	20	**gamdunit Clock Gating Disable** Access: R/W gamdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	19	**vdlunit1 Clock Gating Disable** Access: R/W vdlunit1 Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	18	**vhmeunit Clock Gating Disable** Access: R/W vhmeunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	17	**vcreunit Clock Gating Disable** Access: R/W vcreunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	16	**hleunit Clock Gating Disable** Access: R/W hleunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL8 - Unit Level Clock Gating Control 8

15	**mbdunit Clock Gating Disable** Access: R/W mbdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
14	**mmxunit Clock Gating Disable** Access: R/W mmxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
13	**mpdunit Clock Gating Disable** Access: R/W mpdunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
12	**hedunit Clock Gating Disable** Access: R/W hedunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
11	**hlfunit Clock Gating Disable** Access: R/W hlfunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
10	**hmcunit Clock Gating Disable** Access: R/W hmcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL8 - Unit Level Clock Gating Control 8

	9	**hmxunit Clock Gating Disable** Access: R/W hmxunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	8	**hppunit Clock Gating Disable** Access: R/W hppunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	7	**hprunit Clock Gating Disable** Access: R/W hprunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	6	**Reserved**
	5	**hwmunit Clock Gating Disable** Access: R/W hwmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	4	**mdcunit Clock Gating Disable** Access: R/W mdcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	3	**vmpcunit Clock Gating Disable** Access: R/W vmpcunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

UCGCTL8 - Unit Level Clock Gating Control 8

	2	**sfmunit Clock Gating Disable ebb** Access: R/W sfmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**sfaunit Clock Gating Disable ebb** Access: R/W sfaunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**sfeunit Clock Gating Disable ebb** Access: R/W sfeunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

Unit Level Clock Gating Control 9

UCGCTL9 - Unit Level Clock Gating Control 9

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	0943Ch

Unit Level Clock Gating Disable bits

DWord	Bit	Description
0	31:4	**Reserved** Access: RO Reserved
	3	**vbspunit Clock Gating Disable** Access: R/W vbspunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	2	**vmmunit Clock Gating Disable** Access: R/W vmmunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	1	**AVSunit Clock Gating Disable** Access: R/W AVSunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)
	0	**daprssunit Clock Gating Disable** Access: R/W daprssunit Clock Gating Disable Control: '0' : Clock Gating Enabled. (i.e., clocks can be gated when they are not required to toggle for functionality) '1' : Clock Gating Disabled. (i.e., clocks are toggling, always)

URB Context Offset

URB_CXT_OFFSET - URB Context Offset		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	RenderCS	
Default Value:	0x00009AC0	
Access:	Read/32 bit Write Only	
Size (in bits):	32	
Address:	021B8h	
DWord	**Bit**	**Description**
0	31:6	**URB Offset** Default Value: 26Bh This field indicates the offset (64bytes granular) in to the logical rendering context to which URB contents are save/restored when enabled. This field register must not be written directly (via MMIO) unless the Command Streamer is completely idle (i.e., the Ring Buffer is empty and the pipeline is idle) and RC6 is disabled. One way to program this register is via Load Register Immediate command in the ring buffer as part of initialization sequence.
	5:0	**Reserved** Format: MBZ

Valid Bit Vector 0 for CVS

CVSTLB_VLD_0 - Valid Bit Vector 0 for CVS	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04C00h

This register contains the valid bits for entries 0-31 of CVSTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for CVS** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for L3

L3TLB_VLD_0 - Valid Bit Vector 0 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D00h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for MFX

MFXTLB_VLD_0 - Valid Bit Vector 0 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BA0h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for MFX SL1

MFXTLB_VLD_SL1_0 - Valid Bit Vector 0 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BC0h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for MTTLB

MTTLB_VLD0 - Valid Bit Vector 0 for MTTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04780h-04783h

This register contains the valid bits for entries 0-31 of MTTLB (Texture and constant cache TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 0 for MTVICTLB

VICTLB_VLD0 - Valid Bit Vector 0 for MTVICTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04788h-0478Bh

This register contains the valid bits for entries 0-31 of VICTLB (Vertex Fetch, Instruction Cache, and Command Streamer TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 0 for RCC

RCCTLB_VLD_0 - Valid Bit Vector 0 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DA0h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for RCCTLB

RCCTLB_VLD0 - Valid Bit Vector 0 for RCCTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04790h-04793h

This register contains the valid bits for entries 0-31 of RCCTLB (Render Cache for Color TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 0 for RCZTLB

RCZTLB_VLD0 - Valid Bit Vector 0 for RCZTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04798h-0479Bh
This register contains the valid bits for entries 0-31 of RCZTLB (Render Cache for Z (Depth), Hi Z, and Stencil TLB).	

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 0 for TLBPEND registers

TLBPEND_VLD0 - Valid Bit Vector 0 for TLBPEND registers	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04700h-04703h

This register contains the valid bits for entries 0-31 of TLBPEND structure (Cycles pending TLB translation).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 0 for VEBX

VEBXTLB_VLD_0 - Valid Bit Vector 0 for VEBX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B20h

This register contains the valid bits for entries 0-31 of VEBXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for VEBX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for WIDI

BWDTLB_VLD_0 - Valid Bit Vector 0 for WIDI	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DC0h

This register contains the valid bits for entries 0-31 of BWDTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for WIDI** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 0 for Z

ZTLB_VLD_0 - Valid Bit Vector 0 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B34h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 0 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for CVS

CVSTLB_VLD_1 - Valid Bit Vector 1 for CVS	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04C04h

This register contains the valid bits for entries 0-31 of CVSTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for CVS** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for L3

L3TLB_VLD_1 - Valid Bit Vector 1 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D04h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for MFX

MFXTLB_VLD_1 - Valid Bit Vector 1 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BA4h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for MFX SL1

MFXTLB_VLD_SL1_1 - Valid Bit Vector 1 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BC4h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for MTTLB

MTTLB_VLD1 - Valid Bit Vector 1 for MTTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04784h-04787h

This register contains the valid bits for entries 0-31 of MTTLB (Texture and constant cache TLBVertex Fetch, Instruction Cache, and Command Streamer TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 1 for MTVICTLB

MTVICTLB_VLD1 - Valid Bit Vector 1 for MTVICTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	0478Ch-0478Fh

This register contains the valid bits for entries 0-31 of VICTLB (Vertex Fetch, Instruction Cache, and Command Streamer TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 1 for RCC

RCCTLB_VLD_1 - Valid Bit Vector 1 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DA4h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for RCCTLB

RCCTLB_VLD1 - Valid Bit Vector 1 for RCCTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04794h-04797h

This register is reserved for future RCC TLB extension.

DWord	Bit	Description
0	31:0	**Reserved** Format: MBZ

Valid Bit Vector 1 for RCZTLB

RCZTLB_VLD1 - Valid Bit Vector 1 for RCZTLB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	0479Ch-0479Fh

This register contains the valid bits for entries 0-31 of RCZTLB (Render Cache for Z (Depth), Hi Z, and Stencil TLB).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 1 for TLBPEND registers

TLBPEND_VLD1 - Valid Bit Vector 1 for TLBPEND registers	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	04704h-04707h

This register contains the valid bits for entries 32-63 of TLBPEND structure (Cycles pending TLB translation).

DWord	Bit	Description
0	31:0	**Valid bits per entry**

Valid Bit Vector 1 for VEBX

VEBXTLB_VLD_1 - Valid Bit Vector 1 for VEBX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B24h

This register contains the valid bits for entries 0-31 of VEBXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for VEBX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for WIDI

BWDTLB_VLD_1 - Valid Bit Vector 1 for WIDI	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DC4h

This register contains the valid bits for entries 0-31 of BWDTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for WIDI** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 1 for Z

ZTLB_VLD_1 - Valid Bit Vector 1 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B38h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 1 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for CVS

CVSTLB_VLD_2 - Valid Bit Vector 2 for CVS	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04C08h

This register contains the valid bits for entries 0-31 of CVSTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for CVS** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for GAB

BWDTLB_VLD_3 - Valid Bit Vector 2 for GAB	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DCCh

This register contains the valid bits for entries 0-31 of BWDTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for GAB** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for L3

L3TLB_VLD_2 - Valid Bit Vector 2 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D08h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for MFX

MFXTLB_VLD_2 - Valid Bit Vector 2 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BA8h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for MFX SL1

MFXTLB_VLD_SL1_2 - Valid Bit Vector 2 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BC8h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for RCC

RCCTLB_VLD_2 - Valid Bit Vector 2 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DA8h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 2 for Z

ZTLB_VLD_2 - Valid Bit Vector 2 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B3Ch

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 2 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for CVS

CVSTLB_VLD_3 - Valid Bit Vector 3 for CVS	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04C0Ch

This register contains the valid bits for entries 0-31 of CVSTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for CVS** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for L3

L3TLB_VLD_3 - Valid Bit Vector 3 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D0Ch

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for MFX

MFXTLB_VLD_3 - Valid Bit Vector 3 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BACh

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for MFX SL1

MFXTLB_VLD_SL1_3 - Valid Bit Vector 3 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BCCh

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for RCC

RCCTLB_VLD_3 - Valid Bit Vector 3 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DACh

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 3 for Z

ZTLB_VLD_3 - Valid Bit Vector 3 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B40h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 3 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 4 for L3

L3TLB_VLD_4 - Valid Bit Vector 4 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D10h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 4 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 4 for MFX

MFXTLB_VLD_4 - Valid Bit Vector 4 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BB0h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 4 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 4 for MFX SL1

MFXTLB_VLD_SL1_4 - Valid Bit Vector 4 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BD0h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 4 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 4 for RCC

RCCTLB_VLD_4 - Valid Bit Vector 4 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DB0h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 4 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 4 for Z

ZTLB_VLD_4 - Valid Bit Vector 4 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B44h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 4 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 5 for L3

L3TLB_VLD_5 - Valid Bit Vector 5 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D14h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 5 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 5 for MFX

MFXTLB_VLD_5 - Valid Bit Vector 5 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BB4h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 5 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 5 for MFX SL1

MFXTLB_VLD_SL1_5 - Valid Bit Vector 5 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BD4h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 5 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 5 for RCC

RCCTLB_VLD_5 - Valid Bit Vector 5 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DB4h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 5 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 5 for Z

ZTLB_VLD_5 - Valid Bit Vector 5 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B48h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 5 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 6 for L3

L3TLB_VLD_6 - Valid Bit Vector 6 for L3		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04D18h	
This register contains the valid bits for entries 0-31 of L3TLB.		
DWord	**Bit**	**Description**
0	31:0	**Valid Bit Vector 6 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 6 for MFX

MFXTLB_VLD_6 - Valid Bit Vector 6 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BB8h

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 6 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 6 for MFX SL1

MFXTLB_VLD_SL1_6 - Valid Bit Vector 6 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BD8h

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 6 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 6 for RCC

RCCTLB_VLD_6 - Valid Bit Vector 6 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DB8h

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 6 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 6 for Z

ZTLB_VLD_6 - Valid Bit Vector 6 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B4Ch

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 6 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 7 for L3

L3TLB_VLD_7 - Valid Bit Vector 7 for L3		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04D1Ch	
This register contains the valid bits for entries 0-31 of L3TLB.		

DWord	Bit	Description
0	31:0	**Valid Bit Vector 7 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 7 for MFX

MFXTLB_VLD_7 - Valid Bit Vector 7 for MFX	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BBCh

This register contains the valid bits for entries 0-31 of MFXTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 7 for MFX** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 7 for MFX SL1

MFXTLB_VLD_SL1_7 - Valid Bit Vector 7 for MFX SL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04BDCh

This register contains the valid bits for entries 0-31 of MFX SL1 TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 7 for MFX SL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 7 for RCC

RCCTLB_VLD_7 - Valid Bit Vector 7 for RCC	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04DBCh

This register contains the valid bits for entries 0-31 of RCCTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 7 for RCC** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 7 for Z

ZTLB_VLD_7 - Valid Bit Vector 7 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B50h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 7 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 8 for L3

L3TLB_VLD_8 - Valid Bit Vector 8 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D20h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 8 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 8 for Z

ZTLB_VLD_8 - Valid Bit Vector 8 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B54h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 8 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 9 for L3

L3TLB_VLD_9 - Valid Bit Vector 9 for L3		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04D24h	
This register contains the valid bits for entries 0-31 of L3TLB.		
DWord	**Bit**	**Description**
0	31:0	**Valid Bit Vector 9 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 9 for Z

ZTLB_VLD_9 - Valid Bit Vector 9 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B58h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 9 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 10 for L3

L3TLB_VLD_10 - Valid Bit Vector 10 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D28h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 10 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 10 for Z

ZTLB_VLD_10 - Valid Bit Vector 10 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B5Ch

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 10 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 11 for L3

L3TLB_VLD_11 - Valid Bit Vector 11 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D2Ch

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 11 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 11 for Z

ZTLB_VLD_11 - Valid Bit Vector 11 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B60h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 11 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 12 for L3

L3TLB_VLD_12 - Valid Bit Vector 12 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D30h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 12 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 12 for Z

ZTLB_VLD_12 - Valid Bit Vector 12 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B64h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 12 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 13 for L3

L3TLB_VLD_13 - Valid Bit Vector 13 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D34h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 13 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 13 for Z

ZTLB_VLD_13 - Valid Bit Vector 13 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B68h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 13 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 14 for L3

L3TLB_VLD_14 - Valid Bit Vector 14 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D38h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 14 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 14 for Z

ZTLB_VLD_14 - Valid Bit Vector 14 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B6Ch

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 14 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 15 for L3

L3TLB_VLD_15 - Valid Bit Vector 15 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D3Ch

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 15 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 15 for Z

ZTLB_VLD_15 - Valid Bit Vector 15 for Z	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B70h

This register contains the valid bits for entries 0-31 of ZTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 15 for Z** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 16 for L3

L3TLB_VLD_16 - Valid Bit Vector 16 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D40h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 16 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 17 for L3

L3TLB_VLD_17 - Valid Bit Vector 17 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D44h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 17 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 18 for L3

L3TLB_VLD_18 - Valid Bit Vector 18 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D48h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 18 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 19 for L3

L3TLB_VLD_19 - Valid Bit Vector 19 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D4Ch

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 19 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 20 for L3

L3TLB_VLD_20 - Valid Bit Vector 20 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D50h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 20 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 21 for L3

L3TLB_VLD_21 - Valid Bit Vector 21 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D54h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 21 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 22 for L3

L3TLB_VLD_22 - Valid Bit Vector 22 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D58h

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 22 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector 23 for L3

L3TLB_VLD_23 - Valid Bit Vector 23 for L3	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04D5Ch

This register contains the valid bits for entries 0-31 of L3TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector 23 for L3** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector for VLF

VLFTLB_VLD - Valid Bit Vector for VLF	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B30h

This register contains the valid bits for entries 0-31 of VLFTLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector for VLF** Default Value: 00000000h Access: RO Valid Bits per Entry.

Valid Bit Vector for VLFSL1

VLFSL1TLB_VLD - Valid Bit Vector for VLFSL1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04B2Ch

This register contains the valid bits for entries 0-31 of VLFSL1TLB.

DWord	Bit	Description
0	31:0	**Valid Bit Vector for VLFSL1** Default Value: 00000000h Access: RO Valid Bits per Entry.

VC

VC - VC

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	000B4h

Vendor Capabilities. Any SKU related fuses would be added here.

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 0000000h Access: RO Reserved
	0	**Spare_bit** Default Value: 0b Access: RO Placeholder for SKU related fusing.

VCES Idle Switch Delay

VECS_IDLEDLY - VCES Idle Switch Delay

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A23Ch

The IDLEDLY register contains an Idle Delay field which specifies the minimum number of microseconds allowed for command streamer to wait before a context is switched out leading to IDLE state in execlists mode, i.e following this context switch there is no active element available in HW to execute.
A default value of 0, means that by default, there is no restriction to wait on a context switch leading to IDLE. This register has no significance when execlists are not enabled.

DWord	Bit	Description
0	31:21	**Reserved** Project: All Format: MBZ
	20:0	**IDLE Delay** Default Value: 0h Project: All Format: U21 Minimum number of micro-seconds allowed.

VCID

VCID - VCID

Register Space:	PCI: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01070009
Size (in bits):	32
Address:	000B0h

Vendor Capability ID

DWord	Bit	Description
0	31:24	**VERSION** Default Value: 01h Access: RO VS: Identifies this as the first revision of the CAPID register definition.
	23:16	**LENGTH** Default Value: 07h Access: RO LEN: This field has the value 07h to indicate structure length (8 bytes)
	15:8	**NEXT_CAPABILITY_POINTER** Default Value: 00h Access: R/W Once 00 indicates capability list ends here. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset. Write-once allowing the capability list to be changed.
	7:0	**CAPABILITY_ID_CID** Default Value: 09h Access: RO Identifies this as a vendor dependent capability pointers.

VCS_PREEMPTION_HINT

VCS_PREEMPTION_HINT - VCS_PREEMPTION_HINT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	124BCh
Valid Projects:	CHV, BSW

This register contains the Head pointer offset in to the Ring Buffer or the Dword aligned Graphics address in to the Batch Buffer corresponding to either MI_ARB_CHECK called Preemption Hint Address. When Preemption Hint Address is enabled, VCS will honor UHPTR only on parsing MI_ARB_CHK at Preemption Hint Address.

This register contains the Head pointer offset into the Ring Buffer or the Dword aligned Graphics address into the Batch Buffer corresponding to the below listed commands in execlist mode of operation

- MI_ARB_CHECK
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT

Programming Notes

Programming Restriction: This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHECK in command stream. Programmer has to ensure that VCS Preemption Hint register gets programmed before UHPTR is programmed and well before VCS crosses the corresponding execution point. Preemption hint for both RingBuffer and Batch Buffer can't be enabled simultaneously.

DWord	Bit	Description
0	31:2	**Preempted Hint Address** Format: U30 Format: GraphicsAddress[31:2] This field contains the Head offset in to the Ring Buffer when Preemption Hint is set to Ring Buffer and Dword aligned Graphics Address in to the batch buffer when Preemption Hint is set to Batch Buffer.
	1	**Batch Buffer Preemption Hint** Format: Enable Value / Name / Description 0h / Disabled / Preemption hint is disabled in batch buffer.

VCS_PREEMPTION_HINT - VCS_PREEMPTION_HINT

		1h	Enabled	Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Batch Buffer.
	0	**Ring Preemption Hint**		

Format:	Enable

Value	Name	Description
0h	Disable	Preemption hint is disabled in ring buffer.
1h	Enabled	Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Ring Buffer.

VCS_PREEMPTION_HINT_UDW

VCS_PREEMPTION_HINT_UDW - VCS_PREEMPTION_HINT_UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	124C8h
Valid Projects:	CHV, BSW

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the Batch Buffer corresponding to MI_ARB_CHECK command called Preemption Hint Address.

Programming Notes

Programming Restriction:This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHK in command stream.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Preempted Hint Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer when Preemption Hint is set to Batch Buffer. This field is not valid when Preemption Hint is set to Ring Buffer.

VCS Context ID Preemption Hint

VCS_CTXID_PREEMPTION_HINT - VCS Context ID Preemption Hint

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	124CCh
Valid Projects:	CHV, BSW

This register contains the Context ID of a context in execlist mode of operation. In execlist mode of operation VCS_PREEMPTION_HINT registers are looked at by Video Command Streamer on executing a context having Context ID that matches with the contents of this register. This register contents are valid and looked at only in execlist mode of operation.

Programming Notes

This register should NEVER be programmed in functional mode, this must be used only in validation mode to achieve deterministic preemption behavior in execlist mode of operation.

DWord	Bit	Description
0	31:0	**Context ID Preemption Hint** Format: U32 If 0 this field has no effect. If nonzero it indicates the only context ID that can be preempted when execlists are enabled. A preemption attempt when the context ID of the currently executing ring context does not match this field will be ignored.

VCS Context Sizes

VCS_CXT_SIZE - VCS Context Sizes

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x000A1105 CHV, BSW
Access:	Read/32 bit Write Only
Size (in bits):	32
Address:	121A8h
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:16	**VCS Context Size** Format: U5 Value: Ah, Name: **[Default]**, Project: CHV, BSW
	15:13	**Reserved** Format: MBZ
	12:8	**VCR Context Size** Format: U5 Value: 11h, Name: **[Default]**, Project: CHV, BSW
	7:5	**Reserved** Format: MBZ
	4:0	**Execlist Context Size** Format: U5 Value: 5h, Name: **[Default]**, Project: CHV, BSW

VCS Context Timestamp Count

VCS_CTX_TIMESTAMP - VCS Context Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	123A8h
Valid Projects:	CHV, BSW

This register provides a mechanism to obtain cumulative run time of a GPU context on HW. This register gets context save/restored on a context switch. SW must reset this register on very first submission of a context to HW, then afterwards gets context save/restored maintaining the cumulative run time of the corresponding context. This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register gets reset on an engine reset.

This register is context save restore on a context switch.

DWord	Bit	Description
0	31:0	**Timestamp Value** Format: U32 This register increments for every 80 ns of time.

VCS Counter for the bit stream decode engine

VCS_CNTR - VCS Counter for the bit stream decode engine	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	12178h-1217Bh
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:0	**Count Value** Default Value: ffffffffh Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter.

VCS Error Identity Register

VCS_EIR - VCS Error Identity Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/WC
Size (in bits):	32
Address:	120B0h
Valid Projects:	CHV, BSW

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a 1 to the appropriate bit(s) except for the unrecoverable bits described).

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Error Identity Bits** Format: Array of Error condition bits ee the table titled Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

Value	Name	Description
0h	**[Default]**	
1h	Error occurred	Error occurred

Programming Notes
Writing a 1 to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) cannot be cleared except by reset (i.e., it is a fatal error).

VCS Error Mask Register

VCS_EMR - VCS Error Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0xFFFFFFFF CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	120B4h
Valid Projects:	CHV, BSW

The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.

Undefined or reserved bits in the Hardware Detected Error Bit Table will always return a read value of '1'

DWord	Bit	Description
0	31:16	**Reserved** Default Value: FFFFh Project: CHV, BSW Format: Must Be One
	15:0	**Error Mask Bits** Format: Array of error condition mask bits See the table titled Hardware-Detected Error Bits. This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR. **Value** / **Name** / **Description** 0000h / Not Masked / Will be reported in the EIR FFFFh / Masked **[Default]** / Will not be reported in the EIR

VCS Error Status Register

VCS_ESR - VCS Error Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	120B8h
Valid Projects:	CHV, BSW

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition persistent). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Error Status Bits** Format: Array of error condition bits See the table titled Hardware-Detected Error Bits. This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description
0h	**[Default]**	
1h	Error Condition Detected	Error Condition detected

VCS Execute Condition Code Register

VCS_EXCC - VCS Execute Condition Code Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W, RO
Size (in bits):	32
Trusted Type:	1
Address:	12028h
Valid Projects:	CHV, BSW

This register contains user defined and hardware generated conditions that are used by MI_WAIT_FOR_EVENT commands. An MI_WAIT_FOR_EVENT instruction excludes the executing ring from arbitration if the selected event evaluates to a 1, while instruction is discarded if the condition evaluates to a 0. Once excluded, a ring is enabled into arbitration when the selected condition evaluates to a 0.
This register also contains control for the invalidation of indirect state pointers on context restore.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] These bits serve as a write enable for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15:5	**Reserved** Format: MBZ
	4:0	**Reserved** Project: CHV, BSW Format: MBZ

VCS General Purpose Register

VCS_GPR - VCS General Purpose Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	12600h-1267Fh
Valid Projects:	CHV, BSW

This is a general purpose register bank of sixteen 64-bit registers, used as temporary storage by the MI_MATH command to do ALU operations.

Programming Notes

Any operation that initiates a read to register 0x1266C will return the value of 0x1260c register. This does not include context save or MI_MATH command operation.

DWord	Bit	Description
0	63:0	**Reserved** Format: MBZ

VCS Hardware Status Mask Register

VCS_HWSTAM - VCS Hardware Status Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	12098h
Valid Projects:	CHV, BSW

Access: RO for Reserved Control bits

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

DWord	Bit	Description
0	31:0	**Hardware Status Mask Register** Default Value: FFFFFFFFh Format: Array of Masks Refer to the table in the Interrupt Control Register section for bit definitions.

VCS IDLE Max Count

VCS_PWRCTX_MAXCNT - VCS IDLE Max Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000040 CHV, BSW
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	12054h
Valid Projects:	CHV, BSW

This register contains the time in 0.64us to wait before telling power management hardware the render pipe is IDLE

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ
	19:0	**Reserved**

VCS Idle Switch Delay

VCS_IDLEDLY - VCS Idle Switch Delay

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1223Ch
Valid Projects:	CHV, BSW

The IDLEDLY register contains an Idle Delay field which specifies the minimum number of microseconds allowed for command streamer to wait before a context is switched out leading to IDLE state in Execlist mode, i.e following this context switch there is no active element available in HW to execute.
A default value of 0, means that by default, there is no restriction to wait on a context switch leading to IDLE. This register has no significance when Execlists are not enabled.

DWord	Bit	Description
0	31:21	**Reserved** Project: All Format: MBZ
	20:0	**IDLE Delay** Project: All Format: U21 Minimum number of micro-seconds allowed.

VCS Instruction Parser Mode Register

VCS_INSTPM - VCS Instruction Parser Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	120C0h-120C3h
Valid Projects:	CHV, BSW

The VCS_INSTPM register is used to control the operation of the VCS Instruction Parser. Certain classes of instructions can be disabled (ignored) - often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated - useful for ensuring the completion (vs. only parsing) of rendering instructions. DefaultValue=0000 0000h

Programming Notes

All reserved bits are implemented.

DWord	Bit	Description
0	31:16	**Masks** Format: Mask[15:0] These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15:11	**Reserved** Project: All Format: MBZ
	10	**Reserved**
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Reserved** Format: MBZ
	6:5	**Reserved** Project: CHV, BSW Format: MBZ
	4:0	**Reserved** Access: R/W

VCS_INSTPM - VCS Instruction Parser Mode Register			
		Format:	MBZ

VCS Interrupt Mask Register

VCS_IMR - VCS Interrupt Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	120A8h
Valid Projects:	CHV, BSW

The IMR register is used by software to control which Interrupt Status Register bits are masked or unmasked. Unmasked bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. Masked bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** Format: Array of interrupt mask bits Refer to the Interrupt Control Register section for bit definitions. This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description
FFFF FFFFh	**[Default]**	
0h	Not Masked	Will be reported in the IIR
1h	Masked	Will not be reported in the IIR

VCS Mode Register for Software Interface

VCS_MI_MODE - VCS Mode Register for Software Interface

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000200 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	1209Ch-1209Fh
Valid Projects:	CHV, BSW

The MI_MODE register contains information that controls software interface aspects of the command parser.

DWord	Bit	Description
0	31:16	**Masks** A 1 in a bit in this field allows the modification of the corresponding bit in Bits 15:0.
	15	**Suspend Flush** Mask: MMIO(0x209c)#31 Value / Name / Description: 0h / No Delay / HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well 1h / DelayFlush / Suspend flush is active
	14:12	**Reserved** Access: R/W
	11	**Invalidate UHPTR enable** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
	10	**Atomic Read Return for MI_COPY_MEM_MEM** Project: CHV, BSW Format: U1 Value / Name / Description: 0h / Disable **[Default]** / Hardware does a regular memory fence write to complete the write to the destination address before moving to the next instruction. 1h / Enable / Hardware does Atomic Move with Read Return to complete the write to the destination address before moving to the next instruction.

VCS_MI_MODE - VCS Mode Register for Software Interface

	9	**Ring Idle (Read Only Status bit)** Access: RO *Writes to this bit are not allowed.* 0 – Parser not idle 1 – Parser idle **[Default]**
	8	**Stop Ring** Software must set this bit to force the Ring and Command Parser to Idle. Software must read a "1" in Ring Idle bit after setting this bit to ensure that the hardware is idle. *Software must clear this bit for Ring to resume normal operation.* 0 – Normal Operation 1 – Parser is turned off
	7:0	**Reserved** Access: R/W

VCS Reported Timestamp Count

VCS_TIMESTAMP - VCS Reported Timestamp Count	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000, 0x00000000
Access:	RO. This register is not set by the context restore.
Size (in bits):	64
Address:	12358h
Valid Projects:	CHV, BSW

This register provides an elapsed real-time value that can be used as a timestamp. This register is not reset by a graphics reset. It will maintain its value unless a full chipset reset is performed. Note: This timestamp register reflects the value of the PCU TSC. The PCU TSC counts 10ns increments; this timestamp reflects bits 38:3 of the TSC (i.e. 80ns granularity, rolling over every 1.5 hours).

DWord	Bit	Description
0	63:36	**Reserved** Format: MBZ
	35:0	**Timestamp Value** Format: U36 This register toggles every 80 ns. The upper 28 bits are zero.

VCS Reset Control Register

VCS_RESET_CTRL - VCS Reset Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	120D0h
Valid Projects:	CHV, BSW

This register is to be used to control soft reset.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:2	**Reserved** Format: MBZ
	1	**Ready for Reset** Format: U1 When set indicates video codec engine is ready for reset. This bit gets cleared on engine reset or when Soft Reset In progress is cleared.
	0	**Request Reset** Format: U1 When set indicates SW wishes to reset the video codec engine. On seeing this bit set Command Streamer will take appropriate action and set Ready For Reset status bit. This bit gets cleared on engine reset.

VCS Ring Buffer Next Context ID Register

VCS_RNCID - VCS Ring Buffer Next Context ID Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Address:	12198h-1219Fh
Valid Projects:	CHV, BSW

This register contains the next ring context ID associated with the ring buffer.

Programming Notes

The current context (RCCID) register can be updated indirectly from this register on a context switch event. Note that this can only be triggered when arbitration is enabled or if the current context runs dry (head pointer becomes equal to tail pointer).

DWord	Bit	Description
0	63:0	**Context ID** See Context Descriptor for VCS.

VCS Semaphore Polling Interval on Wait

VCS_SEMA_WAIT_POLL - VCS Semaphore Polling Interval on Wait

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1224Ch
Valid Projects:	CHV, BSW

The SEMA_WAIT_POLL register contains Poll Interval field which specifies the minimum number of microseconds allowed for command streamer to wait before re-fetching the data from the address mentioned in the MI_SEMAPHORE_WAIT command on WAIT Mode set to POLL until the condition is satisfied while the context is not switched out. When value of 0 is written the poll interval will be equal to the memory latency of the read completion.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:0	**Poll Interval** Minimum number of micro-seconds allowed

VCS Threshold for the counter of bit stream decode engine

VCS_THRSH - VCS Threshold for the counter of bit stream decode engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS
Default Value:	0x00150000
Access:	R/W
Size (in bits):	32
Address:	1217Ch-1217Fh

DWord	Bit	Description
0	31:0	**Threshold Value** Default Value: 00150000h The value in this register reflects the number of clocks the bit stream decode engine is expected to run. If the value is exceeded the counter is reset and an interrupt may be enabled in the device.

VCW Clock Count

VCW_CLOCK_CNT - VCW Clock Count			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	VideoEnhancementCS		
Default Value:	0x00000000		
Access:	RO		
Size (in bits):	32		
Trusted Type:	1		
Address:	08820h		
ShortName:	VCW0_CLOCK_CNT		
Address:	08920h		
ShortName:	VCW1_CLOCK_CNT		
DWord	**Bit**	**Description**	
0	31:24	**Reserved** Format:	MBZ
	23:0	**Max clock count** Default Value:	0h Maximum number of clocks taken by VCW to process a column

VCW Internal Latency

VCW_INTERNAL_LAT - VCW Internal Latency		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	VideoEnhancementCS	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Trusted Type:	1	
Address:	08824h	
ShortName:	VCW0_INTERNAL_LAT	
Address:	08924h	
ShortName:	VCW1_INTERNAL_LAT	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Format: MBZ
	23:0	**VCW internal data latency count** Default Value: 0h

VCW Min Max Latency

VCW_MINMAX_LAT - VCW Min Max Latency		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	VideoEnhancementCS	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Trusted Type:	1	
Address:	08828h	
ShortName:	VCW0_MINMAX_LAT	
Address:	08928h	
ShortName:	VCW1_MINMAX_LAT	
DWord	**Bit**	**Description**
0	31:16	**Current request count** Default Value: 0h
	15:8	**Max latency** Default Value: 0h Maximum number of clocks taken for tag 200h
	7:0	**Min latency** Default Value: 0h Minimum number of clocks taken for tag 200h

VCW Total Latency

VCW_TOTAL_LAT - VCW Total Latency	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	0882Ch
ShortName:	VCW0_TOTAL_LAT
Address:	0892Ch
ShortName:	VCW1_TOTAL_LAT

DWord	Bit	Description
0	31:0	**Total latency** Default Value: 0h Accumumation of latency per frame for tag 200h

VCW XY position

VCW_XY_POS - VCW XY position

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	08830h
ShortName:	VCW0_XY_POS
Address:	08930h
ShortName:	VCW1_XY_POS

DWord	Bit	Description
0	31:16	**Current Y value** Default Value: 0h Current Y position of VCW walker
	15:0	**Current X value** Default Value: 0h Current X position of VCW walker

VEBOX TLB Control Register

VTCR - VEBOX TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04270h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

VEBX Context Element Descriptor (High Part)

VEBX_CTX_EDR_H - VEBX Context Element Descriptor (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044C4h

DWord	Bit	Description
0	31:0	**VEBX Context Element Descriptor (High Part)** Default Value: 00000000h Access: R/W

VEBX Context Element Descriptor (Low Part)

VEBX_CTX_EDR_L - VEBX Context Element Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	044C0h

DWord	Bit	Description
0	31:0	**VEBX Context Element Descriptor (Low Part)** Default Value: 00000009h Access: R/W

VEBX Context Element Descriptor (Low Part)

VEBX_CTX_EDR_L - VEBX Context Element Descriptor (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000009
Size (in bits):	32
Address:	044C0h

DWord	Bit	Description
0	31:0	**VEBX Context Element Descriptor** Default Value: 00000009h Access: R/W

VEBX LRA 0

VEBX_LRA_0 - VEBX LRA 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x2F201F00
Size (in bits):	32
Address:	04A80h

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b Access: RO
	29:24	**VEBX LRA1 Max** Default Value: 101111b Access: R/W Maximum value of programmable LRA1.
	23:22	**Reserved** Default Value: 00b Access: RO
	21:16	**VEBX LRA1 Min** Default Value: 100000b Access: R/W Minimum value of programmable LRA1.
	15:14	**Reserved** Access: RO
	13:8	**VEBX LRA0 Max** Default Value: 011111b Access: R/W Maximum value of programmable LRA0.
	7:6	**Reserved** Default Value: 00b Access: RO
	5:0	**VEBXLRA0 Min** Default Value: 000000b Access: R/W Minimum value of programmable LRA0.

VEBX LRA 1

VEBX_LRA_1 - VEBX LRA 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x06003F30
Size (in bits):	32
Address:	04A84h

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b Access: RO
	29:28	**VECS** Default Value: 00b Access: R/W Which LRA should VECS use.
	27:26	**VFW** Default Value: 01b Access: R/W Which LRA should VFW use.
	25:24	**VEO** Default Value: 10b Access: R/W Which LRA should VEO use.
	23:14	**Reserved** Default Value: 0000000000b Access: RO
	13:8	**VEBXLRA2 Max** Default Value: 111111b Access: R/W Minimum value of programmable LRA2.
	7:6	**Reserved** Default Value: 00b Access: RO

VEBX_LRA_1 - VEBX LRA 1		
	5:0	**VEBXLRA2 Min** Default Value: 110000b Access: R/W Minimum value of programmable LRA2.

VEBX PDP0/PML4/PASID Descriptor (High Part)

VEBX_CTX_PDP0_H - VEBX PDP0/PML4/PASID Descriptor (High Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044CCh

DWord	Bit	Description
0	31:0	**VEBX PDP0/PML4/PASID Descriptor (High Part)** Default Value: 00000000h Access: R/W

VEBX PDP0/PML4/PASID Descriptor (Low Part)

VEBX_CTX_PDP0_L - VEBX PDP0/PML4/PASID Descriptor (Low Part)

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044C8h

DWord	Bit	Description
0	31:0	**VEBX PDP0/PML4/PASID Descriptor (Low Part)** Default Value: 00000000h Access: R/W

VEBX PDP1 Descriptor Register (High Part)

VEBX_CTX_PDP1_H - VEBX PDP1 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044D4h

DWord	Bit	Description
0	31:0	**VEBX PDP1 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

VEBX PDP1 Descriptor Register (Low Part)

VEBX_CTX_PDP1_L - VEBX PDP1 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044D0h

DWord	Bit	Description
0	31:0	**VEBX PDP1 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

VEBX PDP2 Descriptor Register (High Part)

VEBX_CTX_PDP2_H - VEBX PDP2 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044DCh

DWord	Bit	Description
0	31:0	**VEBX PDP2 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

VEBX PDP2 Descriptor Register (Low Part)

VEBX_CTX_PDP2_L - VEBX PDP2 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044D8h

DWord	Bit	Description
0	31:0	**VEBX PDP2 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

VEBX PDP3 Descriptor Register (High Part)

VEBX_CTX_PDP3_H - VEBX PDP3 Descriptor Register (High Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044E4h

DWord	Bit	Description
0	31:0	**VEBX PDP3 Descriptor Register (High Part)** Default Value: 00000000h Access: R/W

VEBX PDP3 Descriptor Register (Low Part)

VEBX_CTX_PDP3_L - VEBX PDP3 Descriptor Register (Low Part)	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	044E0h

DWord	Bit	Description
0	31:0	**VEBX PDP3 Descriptor Register (Low Part)** Default Value: 00000000h Access: R/W

VECS_PREEMPTION_HINT

VECS_PREEMPTION_HINT - VECS_PREEMPTION_HINT

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A4BCh

This register contains the Head pointer offset in to the Ring Buffer or the Dword aligned Graphics address in to the Batch Buffer corresponding to either MI_ARB_CHECK called Preemption Hint Address. When Preemption Hint Address is enabled, VECS will honor UHPTR only on parsing MI_ARB_CHK at Preemption Hint Address.

This register contains the Head pointer offset into the Ring Buffer or the Dword aligned Graphics address into the Batch Buffer corresponding to the below listed commands in execlist mode of operation

- MI_ARB_CHECK
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT

Programming Notes

Programming Restriction:This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHECK in command stream. Programmer has to ensure that VECS Preemption Hint register gets programmed before UHPTR is programmed and well before VECS crosses the corresponding execution point. Preemption hint for both RingBuffer and Batch Buffer can't be enabled simultaneously.

DWord	Bit	Description
0	31:2	**Preempted Hint Address** Format: U30 Format: GraphicsAddress[31:2] This field contains the Head offset in to the Ring Buffer when Preemption Hint is set to Ring Buffer and Dword aligned Graphics Address in to the batch buffer when Preemption Hint is set to Batch Buffer.
	1	**Batch Buffer Preemption Hint** Format: Enable 0h Disabled: Preemption hint is disabled in batch buffer. 1h Enabled: Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Batch Buffer.

VECS_PREEMPTION_HINT - VECS_PREEMPTION_HINT

	0	**Ring Preemption Hint** Format: Enable **Value** / **Name** / **Description** 0h / Disable / Preemption hint is disabled in ring buffer. 1h / Enabled / Preemption hint is enabled in ring buffer and preemption hint address corresponds to the instruction in Ring Buffer.

VECS Context ID Preemption Hint

VECS_CTXID_PREEMPTION_HINT - VECS Context ID Preemption Hint

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A4CCh

This register contains the Context ID of a context in Execlist mode of operation. In execlist mode of operation VECS_PREEMPTION_HINT registers are looked at by Video Enhancement Command Streamer on executing a context having Context ID that matches with the contents of this register. This register contents are valid and looked at only in Execlist mode of operation.

Programming Notes

This register should NEVER be programmed in functional mode, this must be used only in validation mode to achieve deterministic preemption behavior in execlist mode of operation.

DWord	Bit	Description
0	31:0	**Context ID Preemption Hint** Format: U32 If 0 this field has no effect. If nonzero it indicates the only context ID that can be preempted when execlists are enabled. A preemption attempt when the context ID of the currently executing ring context does not match this field will be ignored.

VECS Context Timestamp Count

VECS_CTX_TIMESTAMP - VECS Context Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A3A8h

This register provides a mechanism to obtain cumulative run time of a GPU context on HW. This register gets context save/restored on a context switch. SW must reset this register on very first submission of a context to HW, then afterwards gets context save/restored maintaining the cumulative run time of the corresponding context. Note that the value of this register can be obtained in a 3D pipeline-synchronous fashion without a pipeline flush by using the PIPE_CONTROL command. See 3D Geometry Pipeline in the "3D and Media" volume. This register (effectively) counts at a constant frequency by adjusting the increment amount according to the actual reference clock frequency. SW therefore does not need to know the reference clock frequency. This register gets reset on an engine reset.

This register is context save restore on a context switch.

DWord	Bit	Description
0	31:0	**Timestamp Value** Format: U32 This register increments for every 80 ns of time.

VECS Counter for the Video Enhancement Engine

VECS_CNTR - VECS Counter for the Video Enhancement Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	1A178h

DWord	Bit	Description
0	31:0	**Count Value** Default Value: ffffffffh Writing a Zero value to this register starts the counting. Writing a Value of FFFF FFFF to this counter stops the counter.

VECS Error Identity Register

VECS_EIR - VECS Error Identity Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/WC
Size (in bits):	32
Address:	1A0B0h

The EIR register contains the persistent values of Hardware-Detected Error Condition bits. Any bit set in this register will cause the Master Error bit in the ISR to be set. The EIR register is also used by software to clear detected errors (by writing a '1' to the appropriate bit(s) except for the unrecoverable bits described).

DWord	Bit	Description
0	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Error Identity Bits** Project: All Format: Array of Error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. (See **Error! Reference source not found**.). The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register. In order to clear an error condition, software must first clear the error by writing a '1' to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.

Value	Name	Description	Project
0h	**[Default]**		
1h	Error occurred	Error occurred	All

Programming Notes

Writing a '1' to a set bit will cause that error condition to be cleared. However, the Page Table Error bit (Bit 4) can not be cleared except by reset (i.e., it is a fatal error).

VECS Error Mask Register

VECS_EMR - VECS Error Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0xFFFFFFFF CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	1A0B4h

The EMR register is used by software to control which Error Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the EIR, thus setting the Master Error ISR bit and possibly triggering a CPU interrupt, and will persist in the EIR until cleared by software. "Masked" bits will not be reported in the EIR and therefore cannot generate Master Error conditions or CPU interrupts.

Undefined or reserved bits in the Hardware Detected Error Bit Table will always return a read value of '1'

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="2">0</td><td>31:16</td><td>Reserved
<table>
<tr><td>Default Value:</td><td>FFFFh</td></tr>
<tr><td>Project:</td><td>CHV, BSW</td></tr>
<tr><td>Format:</td><td>Must Be One</td></tr>
</table></td></tr>
<tr><td>15:0</td><td>Error Mask Bits
<table>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Format:</td><td>Array of error condition mask bits See Table 1 5. Hardware-Detected Error Bits</td></tr>
</table>
This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.
<table>
<tr><th>Value</th><th>Name</th><th>Description</th></tr>
<tr><td>0000h</td><td>Not Masked</td><td>Will be reported in the EIR</td></tr>
<tr><td>FFFFh</td><td>Masked [Default]</td><td>Will not be reported in the EIR</td></tr>
</table></td></tr>
</table>

VECS Error Status Register

VECS_ESR - VECS Error Status Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Address:	1A0B8h

The ESR register contains the current values of all Hardware-Detected Error condition bits (these are all by definition "persistent"). The EMR register selects which of these error conditions are reported in the persistent EIR (i.e., set bits must be cleared by software) and thereby causing a Master Error interrupt condition to be reported in the ISR.

DWord	Bit	Description
0	31:16	**Reserved** Project: All Format: MBZ
	15:0	**Error Status Bits** Project: All Format: Array of error condition bits See Table 1 5. Hardware-Detected Error Bits This register contains the non-persistent values of all hardware-detected error condition bits.

Value	Name	Description	Project
0h	**[Default]**		
1h	Error Condition Detected	Error Condition detected	All

VECS General Purpose Register

VECS_GPR - VECS General Purpose Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64

Address: 1A600h-1A67Fh

This is a general purpose register bank of sixteen 64-bit registers, used as temporary storage by the MI_MATH command to do ALU operations.

Programming Notes	Project
Any operation that initiates a read to register 0x1A66C will return the value of 0x1A60c register. This does not include context save or MI_MATH command operation.	CHV, BSW

DWord	Bit	Description
0	63:0	**Reserved** Format: MBZ

VECS Hardware Status Mask Register

VECS_HWSTAM - VECS Hardware Status Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	1A098h

Access: RO for Reserved Control bits

The HWSTAM register has the same format as the Interrupt Control Registers. The bits in this register are "mask" bits that prevent the corresponding bits in the Interrupt Status Register from generating a "Hardware Status Write" (PCI write cycle). Any unmasked interrupt bit (HWSTAM bit set to 0) will allow the Interrupt Status Register to be written to the ISR location (within the memory page specified by the Hardware Status Page Address Register) when that Interrupt Status Register bit changes state.

Programming Notes

- To write the interrupt to the HWSP, the corresponding IMR bit must also be clear (enabled).
- At most 1 bit can be unmasked at any given time.

DWord	Bit	Description
0	31:0	**Hardware Status Mask Register** Default Value: FFFFFFFFh Project: All Format: Array of Masks refer to Table 4-4 in Interrupt Control Register section for bit definitions

VECS IDLE Max Count

VECS_PWRCTX_MAXCNT - VECS IDLE Max Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000040
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	1A054h

This register contains the time in 0.64us to wait before telling power management hardware the render pipe is IDLE

DWord	Bit	Description
0	31:20	**Reserved** Project: All Format: MBZ
	19:0	**Reserved**

VECS Instruction Parser Mode Register

VECS_INSTPM - VECS Instruction Parser Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A0C0h-1A0C3h

The VECS_INSTPM register is used to control the operation of the VECS Instruction Parser. Certain classes of instructions can be disabled (ignored) - often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated - useful for ensuring the completion (vs. only parsing) of rendering instructions.

Programming Notes

All reserved bits are implemented

DWord	Bit	Description
0	31:16	**Masks** Format: Mask[15:0] These bits serve as write enables for bits 15:0. If this register is written with any of these bits clear the corresponding bit in the field 15:0 will not be modified. Reading these bits always returns 0s.
	15:11	**Reserved** Project: All Format: MBZ
	10	**Reserved**
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Reserved** Format: MBZ
	6:5	**Reserved** Project: CHV, BSW Format: MBZ
	4:0	**Reserved**

VECS Interrupt Mask Register

VECS_IMR - VECS Interrupt Mask Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0xFFFFFFFF
Access:	R/W
Size (in bits):	32
Address:	1A0A8h

The IMR register is used by software to control which Interrupt Status Register bits are "masked" or "unmasked". "Unmasked" bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. "Masked" bits will not be reported in the IIR and therefore cannot generate CPU interrupts.

DWord	Bit	Description
0	31:0	**Interrupt Mask Bits** Project: All Format: Array of interrupt mask bits Refer to Table 4-4 in Interrupt Control Register section for bit definitions This field contains a bit mask which selects which interrupt bits (from the ISR) are reported in the IIR.

Value	Name	Description	Project
FFFF FFFFh	**[Default]**		
0h	Not Masked	Will be reported in the IIR	All
1h	Masked	Will not be reported in the IIR	All

VECS Mode Register for Software Interface

VECS_MI_MODE - VECS Mode Register for Software Interface

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000200 CHV, BSW
Access:	R/W
Size (in bits):	32
Address:	1A09Ch-1A09Fh

The MI_MODE register contains information that controls software interface aspects of the command parser

DWord	Bit	Description
0	31:16	**Masks** A "1" in a bit in this field allows the modification of the corresponding bit in Bits 15:0
	15	**Suspend Flush** Project: All Mask: MMIO(0x209c)#31 Value / Name / Description: 0h / No Delay / HW will not delay flush, this bit will get cleared by MI_SUSPEND_FLUSH as well 1h / Delay Flush / Suspend flush is active
	14:12	**Reserved** Format: MBZ
	11	**Invalidate UHPTR Enable** If bit set H/W clears the valid bit of BCS_UHPTR (4134h, bit 0) when current active head pointer is equal to UHPTR.
	10	**Atomic Read Return for MI_COPY_MEM_MEM** Project: CHV, BSW Format: U1 Value / Name / Description: 0h / Disable **[Default]** / Hardware does a regular memory fence write to complete the write to the destination address before moving to the next instruction. 1h / Enable / Hardware does Atomic Move with Read Return to complete the write to the destination address before moving to the next instruction.

VECS_MI_MODE - VECS Mode Register for Software Interface

	9	**Ring Idle (Read Only Status bit)** *Writes to this bit are not allowed.* **Value** / **Name** 0 – Parser not idle 1 – Parser idle **[Default]**
	8	**Stop Ring** 0 = Normal Operation. 1 = Parser is turned off. Software must set this bit to force the Ring and Command Parser to Idle. Software must read a "1" in Ring Idle bit after setting this bit to ensure that the hardware is idle. *Software must clear this bit for Ring to resume normal operation.*
	7:0	**Reserved** Format: MBZ

VECS PREEMPTION HINT UDW

VECS_PREEMPTION_HINT_UDW - VECS PREEMPTION HINT UDW

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A4C8h

This register contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the Batch Buffer corresponding to MI_ARB_CHECK command called Preemption Hint Address.

Programming Notes

Programming Restriction:This register should NEVER be programmed in functional mode, this should be used only in validation mode to achieve deterministic behavior of UHPTR being sampled by a given MI_ARB_CHK in command stream.

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Preempted Hint Address Upper DWORD** Format: GraphicsAddress[47:32] This field contains the 4GB aligned base address of gfx 4GB virtual address space within the host's 64-bit virtual address space of the batch buffer when Preemption Hint is set to Batch Buffer. This field is not valid when Preemption Hint is set to Ring Buffer.

VECS Reported Timestamp Count

VECS_TIMESTAMP - VECS Reported Timestamp Count

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000, 0x00000000
Access:	RO. This register is not set by the context restore.
Size (in bits):	64
Address:	1A358h

This register provides an elapsed real-time value that can be used as a timestamp. This register is *not* reset by a graphics reset. It will maintain its value unless a full chipset reset is performed. Note: This timestamp register reflects the value of the PCU TSC. The PCU TSC counts 10ns increments; this timestamp reflects bits 38:3 of the TSC (i.e. 80ns granularity, rolling over every 1.5 hours).

DWord	Bit	Description
0	63:36	**Reserved** Project: All Format: MBZ
	35:0	**TimeStampValue** Project: All Format: U36 This register toggles every 80 ns. The upper 28 bits are zero.

VECS Reset Control Register

VECS_RESET_CTRL - VECS Reset Control Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A0D0h

This register is to be used to control soft reset.

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15:2	**Reserved** Format: MBZ
	1	**Ready for Reset** Format: U1 When set indicates video enhancement engine is ready for reset. This bit gets cleared on engine reset or when Soft Reset In progress is cleared.
	0	**Request Reset** Format: U1 When set indicates SW wishes to reset the video enhancement engine. On seeing this bit set Command Streamer will take appropriate action and set Ready For Reset status bit. This bit gets cleared on engine reset.

VECS Semaphore Polling Interval on Wait

VECS_SEMA_WAIT_POLL - VECS Semaphore Polling Interval on Wait

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	1A24Ch

The SEMA_WAIT_POLL register contains Poll Interval field which specifies the minimum number of microseconds allowed for command streamer to wait before re-fetching the data from the address mentioned in the MI_SEMAPHORE_WAIT command on WAIT Mode set to POLL until the condition is satisfied while the context is not switched out. When value of 0 is written the poll interval will be equal to the memory latency of the read completion.

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20:0	**Poll Interval** Minimum number of micro-seconds allowed

VECS Threshold for the Counter of Video Enhancement Engine

VECS_CTR_THRSH - VECS Threshold for the Counter of Video Enhancement Engine

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00150000
Access:	R/W
Size (in bits):	32
Address:	1A17Ch

DWord	Bit	Description
0	31:0	**Threshold Value** Default Value: 00150000h The value in this register reflects the number of clocks the bit stream decode engine is expected to run. If the value is exceeded the counter is reset and an interrupt may be enabled in the device.

VEO Current Pipe 0 XY Register

VEO_CURRENT0_XY - VEO Current Pipe 0 XY Register	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	08854h
Address:	08954h

DWord	Bit	Description
0	31:30	**Reserved**
	29:16	**Current Input Pipe 0 X** Default Value: 0h
	15	**Reserved**
	14:0	**Current Input Pipe 0 Y** Default Value: 0h

VEO DN Pipe 0 XY Register

VEO_DN0_XY - VEO DN Pipe 0 XY Register			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	VideoEnhancementCS		
Default Value:	0x00000000		
Access:	RO		
Size (in bits):	32		
Trusted Type:	1		
Address:	0884Ch		
Address:	0894Ch		
DWord	**Bit**	**Description**	
0	31:30	**Reserved**	
	29:16	**DN Pipe 0 X**	
		Default Value:	0h
		dn_input_x[13:0]	
	15	**Reserved**	
	14:0	**DN Pipe 0 Y**	
		Default Value:	0h
		dn_input_y[14:0]	

VEO DN Pipe 1 XY Register

VEO_DN1_XY - VEO DN Pipe 1 XY Register			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	VideoEnhancementCS		
Default Value:	0x00000000		
Access:	RO		
Size (in bits):	32		
Trusted Type:	1		
Address:	08850h		
Address:	08950h		
DWord	**Bit**	**Description**	
0	31:30	**Reserved**	
	29:16	**DN Pipe 1 X**	
		Default Value:	0h
	15	**Reserved**	
	14:0	**DN Pipe 1 Y**	
		Default Value:	0h

VEO DV Count Register

VEO_DV_COUNT - VEO DV Count Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	VideoEnhancementCS	
Default Value:	0x00000000	
Access:	RO	
Size (in bits):	32	
Trusted Type:	1	
Address:	08844h	
Address:	08944h	
DWord	**Bit**	**Description**
0	31:24	**Pipe1 MotinoDV/Hold Maxcount** Default Value: 0h
	23:16	**Pipe1 Pixel History DV/Hold Maxcount** Default Value: 0h
	15:8	**Pipe0 Motion History DV/Hold Maxcount** Default Value: 0h
	7:0	**Pipe0 Pixel History DV/Hold Maxcount** Default Value: 0h

VEO DV Hold Register

VEO_DVHOLD - VEO DV Hold Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	0885Ch
Address:	0895Ch

Datavalid/Hold signals for VEO interface

DWord	Bit	Description
0	31	**vdn_p0_veo_pixel_dv** Default Value: 0h
	30	**veo_vdn_p0_pixel_hold** Default Value: 0h
	29	**vdn_p0_veo_mh_dv** Default Value: 0h
	28	**veo_vdn_p0_mh_hold** Default Value: 0h
	27	**vdn_p0_veo_bne_luma_dv** Default Value: 0h
	26	**veo_vdn_p0_bne_luma_hold** Default Value: 0h
	25	**vdn_p0_veo_bne_chroma_dv** Default Value: 0h
	24	**veo_vdn_p0_bne_chroma_hold** Default Value: 0h
	23	**vdi_p0_veo_pixel_dv** Default Value: 0h
	22	**veo_vdi_p0_pixel_hold** Default Value: 0h
	21	**vdi_p0_veo_stmm_dv** Default Value: 0h

VEO_DVHOLD - VEO DV Hold Register

	20	**veo_vdi_p0_stmm_hold** Default Value: 0h
	19	**vdi_p0_veo_fmd_dv** Default Value: 0h
	18	**veo_vdi_p0_fmd_hold** Default Value: 0h
	17	**Reserved** Default Value: 0h
	16	**Reserved** Default Value: 0h
	15	**vdn_p1_veo_pixel_dv** Default Value: 0h
	14	**veo_vdn_p1_pixel_hold** Default Value: 0h
	13	**vdn_p1_veo_mh_dv** Default Value: 0h
	12	**veo_vdn_p1_mh_hold** Default Value: 0h
	11	**vdn_p1_veo_bne_luma_dv** Default Value: 0h
	10	**veo_vdn_p1_bne_luma_hold** Default Value: 0h
	9	**vdn_p1_veo_bne_chroma_dv** Default Value: 0h
	8	**veo_vdn_p1_bne_chroma_hold** Default Value: 0h
	7	**vdi_p1_veo_pixel_dv** Default Value: 0h
	6	**veo_vdi_p1_pixel_hold** Default Value: 0h
	5	**vdi_p1_veo_stmm_dv** Default Value: 0h
	4	**veo_vdi_p1_stmm_hold** Default Value: 0h
	3	**vdi_p1_veo_fmd_dv**

VEO_DVHOLD - VEO DV Hold Register

		Default Value: 0h
	2	**veo_vdi_p1_fmd_hold** Default Value: 0h
	1	**Reserved** Default Value: 0h
	0	**Reserved** Default Value: 0h

VEO Previous Pipe 0 XY Register

VEO_PREVIOUS0_XY - VEO Previous Pipe 0 XY Register			
Register Space:	MMIO: 0/2/0		
Project:	CHV, BSW		
Source:	VideoEnhancementCS		
Default Value:	0x00000000		
Access:	RO		
Size (in bits):	32		
Trusted Type:	1		
Address:	08858h		
Address:	08958h		
DWord	**Bit**	**Description**	
0	31:30	**Reserved**	
	29:16	**Previous Input Pipe 0 X**	
		Default Value:	0h
	15	**Reserved**	
	14:0	**Previous Input Pipe 0 Y**	
		Default Value:	0h

VF Scratch Pad

VFSKPD - VF Scratch Pad

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Address:	02470h
Address:	083A8h-083ABh
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All bits implemented)
	15	**Reserved**
	14:12	**Reserved**
	11	**Reserved**
	10	**Reserved**
	9	**Reserved**
	8	**End Offset Guardband Disable** Project: CHV, BSW Format: U1 Value / Name / Description: 0h / Enable **[Default]** / When 3DPRIMITIVE.End Offset Enable is set to 1**,** VF will use the guardband to determine when a draw call can be pre-empted. VF will not allow pre-emption in the guardband region. 1h / Disable / When set to Disable**,** there will not be any preemption or GB consideration for autodraw (3DPRIMITIVE.End Offset Enable set to 1).
	7	**Guardband Disable** Format: U1 Value / Name / Description: 0h / Enable **[Default]** / VF will use the guardband to determine when a draw call can be pre-empted. VF will not allow pre-emption in the guardband region. 1h / Disable / VF will not use the guardband to determine when a draw call can be pre-empted. VF will allow pre-emption on any vertex in the draw call.

VFSKPD - VF Scratch Pad

	6	**Reserved**
	5	**TLB Prefectch Enable** Project: CHV, BSW Format: U1 Value 0h: Name: Disable **[Default]**; Description: The VF will generate prefetch of TLB when it is fetching sequential vertex data and four or fewer vertex buffers are valid. Value 1h: Name: Enable; Description: VF will disable prefetch of TLB entries.
	4	**Reserved**
	3	**Reserved**
	2	**Vertex Cache Implicit Disable Inhibit** Format: U1 Value 0h: Name: **[Default]**; Description: Allow VF to disable VS0 when Sequential index or Prim ID is a valid Element. Value 1h: Name: ; Description: VF never implicitly disables the vertex cache. Software must disable the VS0 Cache when required.
	1	**Disable Over Fetch Cache** Project: CHV, BSW Value 0h: Name: **[Default]**; Description: Cache will check for data in cache before making a request to memory Value 1h: Name: ; Description: Always re-fetch new data from memory. **Programming Notes** Note that the Disable Multiple Miss Read squash bit must be cleared for Disable Over Fetch Cache to be set.
	0	**Disable Multiple Miss Read squash** Project: CHV, BSW Format: Disable Value 0h: Name: **[Default]**; Description: Allow VF to squash reads that are to the same cacheline for vertex buffer requests. Value 1h: Name: ; Description: Disallow VF from squashing reads that are to the same cacheline for vertex buffer requests.

VFW Credit Count Register

VFW_CREDIT_CNT - VFW Credit Count Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	VideoEnhancementCS	
Default Value:	0x00000002 CHV, BSW	
Access:	RO	
Size (in bits):	32	
Trusted Type:	1	
Address:	08810h	
Address:	08910h	

DWord	Bit	Description
0	31:8	**Reserved**
	7:0	**Credit Count** The number of outstanding credits between VFW and GAV. If zero VEBOX cannot proceed due to GAV not releasing credits.

Value	Name	Project
2h	**[Default]**	CHV, BSW

VIC Virtual page Address Registers

VICTLB_VA - VIC Virtual page Address Registers

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000
Access:	RO
Size (in bits):	32
Trusted Type:	1
Address:	04900h-04903h

These registers are directly mapped to the current Virtual Addresses in the VICTLB (Vertex Fetch, Instruction Cache, and Command Streamer TLB.)

DWord	Bit	Description
0	31:12	**Address** Format: GraphicsAddress[31:12] Page virtual address.
	11:0	**Reserved** Format: MBZ

VIDEOBUSYCOUNTER

VIDEOBUSYCOUNTER - VIDEOBUSYCOUNTER

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	138124h

SOXi Context Save/Restore : No
The 40-bit HW counter will wrap around. The only clear condition is CZ reset.
When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, its 39:8 of the 40-bit counter are reported.
The units are CZ clock cycles.
It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for those registers for power characterization.
0x13_8104[7] controls if this register should count or if it should be gated: 0= clear, 1= count

DWord	Bit	Description
0	31:0	**VIDEOBUSYTIME** Default Value: 00000000h Access: RO Render RC0 Residency Counter.

Video Enhancement Mode Register

VEBOX_MODE - Video Enhancement Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoEnhancementCS
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	1A29Ch

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**Execlist Enable** Default Value: 0h Project: CHV, BSW Mask: MMIO#31 When set, software can utilize the execlist registers to load a context into hardware. When this bit is clear the execlist mechanism cannot be used. The ring must be loaded via MMIO access. **Programming Notes** This bit is not intended to be changed dynamically. Changing the value of this bit while rendering is in progress will have UNDEFINED results. This bit should be changed only after a full reset and before submitting any commands to the device
	14	**Interrupt Steering Bit** Project: CHV, BSW Format: U1 When set, Command Streamer sends interrupt messages to the SHIM of the ON CHIP Micro Controller through message channel. When reset, Command Streamer sends the interrupt messages to Display Engine as config writes on GAM interface.
	13:10	**Reserved** Project: All Format: MBZ
	9	**Per-Process GTT Enable** Project: All Format: Enable Per-Process GTT BS Mode Enable

VEBOX_MODE - Video Enhancement Mode Register

Value	Name	Description
0h	PPGTT Disable **[Default]**	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space in Basic Scheduler Mode.
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.

Programming Notes	Project
This bit is used for enabling PPGTT access in Ring Buffer mode of scheduling. Privilege field in context descriptor states the same in Execlist mode of scheduling. This field should be set before programming PDP0/1/2/3 registers in order to set the PPGTT translation of memory access.	CHV, BSW

8 **Reserved**

Project:	CHV, BSW

7 **64Bit Virtual Addressing Enable**

Project:	CHV, BSW
Format:	Enable

Per-Process GTT Enable

Value	Name	Description	Project
0h	64Bit Virtual Addressing Disable **[Default]**	When clear indicates GFX operating in 32bit Virtual Addressing for PPGTT based memory access.	All

Programming Notes
This bit is only valid when PPGTT is enabled in ring buffer mode of scheduling. Context Descriptor has a similar bit to control 64bit virtual addressing in execlist mode of scheduling. Irrespective of this field set or clear virtual addresses translated through GGTT are always 32Bit. This field should be programmed before enabling PPGTT access. When this field is not set or for GGTT virtual addresses, Graphics Address [47:32] field of any commands or register exercised by SW should be programmed to 0x0.
64 Bit Virtual Addressing is not supported on CHV, BSW and must not be set.

6:5 **Reserved**

Project:	All
Format:	MBZ

4 **Reserved**

Project:	CHV, BSW

3:1 **Reserved**

Project:	CHV, BSW
Format:	MBZ

VEBOX_MODE - Video Enhancement Mode Register

	0	**Privilege Check Disable** Project: CHV, BSW Format: Enable This field when set, disables Privilege Violation checks on non-privileged batch buffers. When set Privileged commands are allowed to be executed from non-privileged batch buffers.

Video Mode Register

MFX_MODE - Video Mode Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	VideoCS, VideoCS2
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32
Trusted Type:	1
Address:	1229Ch
Valid Projects:	CHV, BSW

DWord	Bit	Description
0	31:16	**Mask Bits** Format: Mask[15:0] Must be set to modify corresponding bit in Bits 15:0. (All implemented bits)
	15	**Execlist Enable** Default Value: 0h Project: CHV, BSW Mask: MMIO#31 When set, software can utilize the execlist registers to load a context into hardware. When this bit is clear the Execution List mechanism cannot be used. The ring must be loaded via MMIO access. **Programming Notes** This bit is not intended to be changed dynamically. Changing the value of this bit while rendering is in progress will have UNDEFINED results. This bit should be changed only after a full reset and before submitting any commands to the device
	14	**Interrupt Steering Bit** Project: CHV, BSW Format: U1 When set, Command Streamer sends interrupt messages to the SHIM of the ON CHIP Micro Controller through message channel. When reset, Command Streamer sends the interrupt messages to Display Engine as config writes on GAM interface.
	13:10	**Reserved** Project: All Format: MBZ

MFX_MODE - Video Mode Register

9	**Per-Process GTT Enable**

Format:	Enable Per-Process GTT BS Mode Enable

Value	Name	Description
0h	PPGTT Disable **[Default]**	When clear, the Global GTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.
1h	PPGTT Enable	When set, the PPGTT will be used to translate memory access from designated commands and for commands that select the PPGTT as their translation space.

Programming Notes
This bit is used for enabling PPGTT access in Ring Buffer mode of scheduling. Privilege field in context descriptor states the same in Execlist Mode of scheduling. This field should be set before programming PDP0/1/2/3 registers in order to set the PPGTT translation of memory access.

8	**Reserved**

Project:	CHV, BSW

7	**64Bit Virtual Addressing Enable**

Project:	CHV, BSW
Format:	Enable

Per-Process GTT Enable

Value	Name	Description
0h	64Bit Virtual Addressing Disable **[Default]**	When clear indicates GFX operating in 32bit Virtual Addressing for PPGTT based memory access.

Programming Notes
This bit is only valid when PPGTT is enabled in ring buffer mode of scheduling. Context Descriptor has a similar bit to control 64bit virtual addressing in execlist mode of scheduling. Irrespective of this field set or clear virtual addresses translated through GGTT are always 32Bit. This field should be programmed before enabling PPGTT access. When this field is not set or for GGTT virtual addresses, Graphics Address [47:32] field of any commands or register exercised by SW should be programmed to 0x0.
64 Bit Virtual Addressing is not supported on CHV, BSW and must not be set.

6:5	**Reserved**

Project:	All
Format:	MBZ

4	**Reserved**

Project:	CHV, BSW

MFX_MODE - Video Mode Register

	3:1	**Reserved** Project: CHV, BSW Format: MBZ
	0	**Privilege Check Disable** Project: CHV, BSW Format: Enable This field when set, disables Privilege Violation checks on non-privileged batch buffers. When set Privileged commands are allowed to be executed from non-privileged batch buffers.

VS Invocation Counter

VS_INVOCATION_COUNT - VS Invocation Counter	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	RenderCS
Default Value:	0x00000000, 0x00000000
Access:	R/W
Size (in bits):	64
Trusted Type:	1
Address:	02320h
Valid Projects:	

This register stores the value of the vertex count shaded by VS. This register is part of the context save and restore.

DWord	Bit	Description
0	63:32	**VS Invocation Count Report UDW** Number of vertices that are dispatched as threads by the VS stage. Updated only when **Statistics Enable** is set in VS_STATE (see the Vertex Shader Chapter in the 3D Volume.)
	31:0	**VS Invocation Count Report LDW** Number of vertices that are dispatched as threads by the VS stage. Updated only when **Statistics Enable** is set in VS_STATE (see the Vertex Shader Chapter in the 3D Volume.)

Wait For Event and Display Flip Flags Register

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	022D0h
Name:	RCS Wait For Event and Display Flip Flags Register
ShortName:	RCS_SYNC_FLIP_STATUS
Valid Projects:	CHV, BSW

Address:	122D0h-122D3h
Name:	Wait For Event and Display Flip Flags Register
ShortName:	SYNC_FLIP_STATUS_VCSUNIT0

Address:	1A2D0h-1A2D3h
Name:	Wait For Event and Display Flip Flags Register
ShortName:	SYNC_FLIP_STATUS_VECSUNIT

Address:	1C2D0h-1C2D3h
Name:	Wait For Event and Display Flip Flags Register
ShortName:	SYNC_FLIP_STATUS_VCSUNIT1

Address:	222D0h
Name:	BCS Wait For Event and Display Flip Flags Register
ShortName:	BCS_SYNC_FLIP_STATUS
Valid Projects:	CHV, BSW

This register is the saved value of what wait for events are still valid. This register is part of context save and restore for RC6 feature.

Programming Notes

Programming Restriction: This register should NEVER be programmed by SW, this is for HW internal use only.

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30	**Display Plane A Asyncronous Display Flip Pending** Format: Enable

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

		This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	29	**Display Plane A Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Plane A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	28	**Display Sprite A Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Sprite A "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	27	**Reserved** Format: MBZ
	26	**Display Plane B Asyncronous Display Flip Pending** Format: Enable This field enables a wait for the duration of a Display Plane B "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	25	**Display Plane B Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	24	**Display Sprite B Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Sprite B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

	23	**Reserved** Source: BlitterCS Format: MBZ
	23	**Display Plane A Asyncronous Performance Flip Pending Wait Enable** Source: RenderCS Format: Enable This field enables a wait for the duration of a Display Plane A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	22	**Display Plane A Asyncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	21	**Display Plane A Syncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	20	**Display Sprite A Syncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Sprite A Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	19	**Reserved** Format: MBZ
	18	**Display Pipe A Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Pipe A Scan Line condition exists. This condition is defined as the the start of the scan line specified in the Pipe A Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

	17	**Display Pipe A Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Pipe A Vertical Blank event occurs. This event is defined as the start of the next Display Pipe A vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See Programming Interface).
	16	**Reserved** Project: CHV, BSW Format: MBZ
	15	**Reserved** Source: BlitterCS Format: MBZ
	15	**Display Plane B Asyncronous Performance Flip Pending Wait Enable** Source: RenderCS Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	14	**Display Plane B Asyncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	13	**Display Plane B Syncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	12	**Display Sprite B Syncronous Flip Pending Wait Enable** Format: Enable

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

		This field enables a wait for the duration of a Display Sprite B Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	11	**Reserved** Format: MBZ
	10	**Display Pipe B Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Pipe B Scan Line condition exists. This condition is defined as the the start of the scan line specified in the Pipe B Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of MI Functions.
	9	**Display Pipe B Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Pipe B Vertical Blank event occurs. This event is defined as the start of the next Display Pipe B vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See Programming Interface).
	8	**Reserved** Project: CHV, BSW Format: MBZ
	7:5	**Reserved** Format: MBZ
	4:0	**Reserved** Project: CHV, BSW Format: MBZ

Wait For Event and Display Flip Flags Register 1

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

Address:	022D4h
Name:	RCS Wait For Event and Display Flip Flags Register 1
ShortName:	RCS_SYNC_FLIP_STATUS_1

Address:	122D4h-122D7h
Name:	Wait For Event and Display Flip Flags Register 1
ShortName:	SYNC_FLIP_STATUS_1_VCSUNIT0

Address:	1A2D4h-1A2D7h
Name:	Wait For Event and Display Flip Flags Register 1
ShortName:	SYNC_FLIP_STATUS_1_VECSUNIT

Address:	1C2D4h-1C2D7h
Name:	Wait For Event and Display Flip Flags Register 1
ShortName:	SYNC_FLIP_STATUS_1_VCSUNIT1

Address:	222D4h
Name:	BCS Wait For Event and Display Flip Flags Register 1
ShortName:	BCS_SYNC_FLIP_STATUS_1
Valid Projects:	CHV, BSW

This register is the saved value of what wait for events are still valid. This register is part of context save and restore for RC6 feature.

DWord	Bit	Description
0	31:27	**Reserved**
	26	**Display Sprite C3 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite C3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

	25	**Display Sprite C3 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite C3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	24	**Display Sprite B3 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite B3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	23	**Display Sprite B3 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite B3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	22	**Display Sprite A3 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite A3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	21	**Display Sprite A3 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite A3 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

	20	**Display Sprite C2 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite C2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	19	**Display Sprite C2 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite C2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	18	**Display Sprite B2 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite B2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	17	**Display Sprite B2 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite B2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	16	**Display Sprite A2 Synchronous Flip Pending Wait Enable** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite A2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

	15	**Display Sprite A2 Synchronous Flip Display Pending** Project: CHV, BSW Format: Enable This field enables a wait for the duration of a Display Sprite A2 Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	14	**Display Plane C Scan Line Event Pending** Project: CHV, BSW Format: Enable This field indicates scan line event operation is pending from Display Plane-C. This field gets set when MI_SCANLINE_INCL/EXCL command is parsed for Display Plane-C and gets reset on scan line event completion for Display Plane-C.
	13	**Display Plane B Scan Line Event Pending** Project: CHV, BSW Format: Enable This field indicates scan line event operation is pending from Display Plane-B. This field gets set when MI_SCANLINE_INCL/EXCL command is parsed for Display Plane-B and gets reset on scan line event completion for Display Plane-B.
	12	**Display Plane A Scan Line Event Pending** Project: CHV, BSW Format: Enable This field indicates scan line event operation is pending from Display Plane-A. This field gets set when MI_SCANLINE_INCL/EXCL command is parsed for Display Plane-A and gets reset on scan line event completion for Display Plane-A.
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10	**Display Plane C Asyncronous Display Flip Pending** Format: Enable This field enables a wait for the duration of a Display Plane C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

	9	**Display Plane C Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Plane C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	8	**Display Sprite C Syncronous Flip Display Pending** Format: Enable This field enables a wait for the duration of a Display Sprite C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	7	**Reserved** Source: BlitterCS Format: MBZ
	7	**Display Plane C Asyncronous Performance Flip Pending Wait Enable** Source: RenderCS Format: Enable This field enables a wait for the duration of a Display Plane C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	6	**Display Plane C Asyncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane C "Flip Pending" condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.
	5	**Display Plane C Syncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Plane C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition (in the Device Programming Interface chapter of MI Functions.

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

	4	**Display Sprite C Syncronous Flip Pending Wait Enable** Format: Enable This field enables a wait for the duration of a Display Sprite C Flip Pending condition. If a flip request is pending, the parser will wait until the flip operation has completed (i.e., the new front buffer address has now been loaded into the active front buffer registers). See Display Flip Pending Condition in the Device Programming Interface chapter of MI Functions.
	3	**Reserved** Format: MBZ
	2	**Display Pipe C Scan Line Wait Enable** Format: Enable This field enables a wait while a Display Pipe C Scan Line condition exists. This condition is defined as the the start of the scan line specified in the Pipe C Display Scan Line Count Range Compare Register. See Scan Line Event in the Device Programming Interface chapter of MI Functions.
	1	**Display Pipe C Vertical Blank Wait Enable** Format: Enable This field enables a wait until the next Display Pipe C Vertical Blank event occurs. This event is defined as the start of the next Display Pipe C vertical blank period. Note that this can cause a wait for up to an entire refresh period. See Vertical Blank Event (See Programming Interface).
	0	**Reserved** Project: CHV, BSW Format: MBZ

Walkers Fault Register

WF_REG - Walkers Fault Register		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04098h	

DWord	Bit	Description
0	31:1	**Walkers Fault Register** Default Value: 00000000000000000000000000000000b Access: R/W All bits are only valid with bit[0]=1.
	0	**Valid Bit** Default Value: 0b Access: R/W This bit indicates that the first fault for this engine has been recorded. It can only be cleared by SW, which also clears the other fields.

WD_WNIC_MSG_ADDR

WD_WNIC_MSG_ADDR - WD_WNIC_MSG_ADDR	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	6E530h

WNIC Message Address. This is a register within Gunit (CZ domain).
Address uses the address as shown.

DWord	Bit	Description
0	31:0	**Reserved**

WGBOX State Arbitration Priority Control

APC - WGBOX State Arbitration Priority Control

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Access:	R/W
Size (in bits):	32

DWord	Bit	Description
0	31:9	**Reserved** Format: MBZ

WIDI LRA 0

WIDI_LRA_0 - WIDI LRA 0

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x37201F00
Size (in bits):	32
Address:	04A90h

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b
	29:24	**WIDI LRA1 Max** Default Value: 110111b Access: R/W Maximum value of programmable LRA1.
	23:22	**Reserved** Default Value: 00b Access: RO
	21:16	**WIDI LRA1 Min** Default Value: 100000b Access: R/W Minimum value of programmable LRA1.
	15:14	**Reserved** Default Value: 00b Access: RO
	13:8	**WIDI LRA0 Max** Default Value: 011111b Access: R/W Maximum value of programmable LRA0.
	7:6	**Reserved** Default Value: 00b Access: RO
	5:0	**WIDILRA0 Min** Default Value: 000000b Access: R/W Minimum value of programmable LRA0.

WIDI LRA 1

WIDI_LRA_1 - WIDI LRA 1	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01000000
Size (in bits):	32
Address:	04A94h

DWord	Bit	Description
0	31:30	**Reserved** Default Value: 00b Access: RO
	29:28	**VMX** Default Value: 00b Access: R/W Which LRA should VMX use.
	27:26	**BSP** Default Value: 00b Access: R/W Which LRA should BSP use.
	25:24	**IME** Default Value: 01b Access: R/W Which LRA should IME/WRS use.
	23:14	**Reserved** Default Value: 0000000000b Access: RO
	13:8	**Reserved**
	7:6	**Reserved** Default Value: 00b Access: RO
	5:0	**Reserved**

WIDI TLB Control Register

WTCR - WIDI TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04278h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

WRID_VALID_REG0

WRID_VALID_REG0 - WRID_VALID_REG0		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04070h	

DWord	Bit	Description
0	31:0	**WRID_VALID_REG0** Default Value: 00000000h Access: RO This register is for WRID Comparison usage. RO register with IA Access Type on DEV reset. wrdp_wrid_valid_vector[31:0] There are 96 write buffer. Each bit indicate the buffer is valid if set. Divide into 3 registers to accommodate all 96 deep

WRID_VALID_REG1

WRID_VALID_REG1 - WRID_VALID_REG1		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04074h	
DWord	**Bit**	**Description**
0	31:0	**WRID_VALID_REG1** Default Value: 00000000h Access: RO This register is for WRID Comparison usage. RO register with IA Access Type on DEV reset. wrdp_wrid_valid_vector[63:32] There are 96 write buffer. Each bit indicate the buffer is valid if set. Divide into 3 registers to accommodate all 96 deep.

WRID_VALID_REG2

WRID_VALID_REG2 - WRID_VALID_REG2		
Register Space:	MMIO: 0/2/0	
Project:	CHV, BSW	
Source:	PRM	
Default Value:	0x00000000	
Size (in bits):	32	
Address:	04078h	
DWord	**Bit**	**Description**
0	31:0	**WRID_VALID_REG2** Default Value: 00000000h Access: RO This register is for WRID Comparison usage. RO register with IA Access Type on DEV reset. wrdp_wrid_valid_vector[95:64] There are 96 write buffer. Each bit indicate the buffer is valid if set. Divide into 3 registers to accommodate all 96 deep.

Write Watermark

WR_WATERMARK - Write Watermark

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x000FFEA4
Size (in bits):	32
Address:	04028h

DWord	Bit	Description
0	31:20	**Extra Bits** Default Value: 000000000000b Access: R/W
	19	**Watermark Timeout Enable** Default Value: 1b Access: R/W
	18:8	**Watermark Timeout** Default Value: 11111111110b Access: R/W Number of clocks that the write pipe queue is allowed to keep a ready write cycle, without reads or writes to the queue. Once this value is met, and if the feature is enabled, the watermark is considered reached, and all pending write requests are issued.
	7	**Watermark Enable** Default Value: 1b Access: R/W Enable Write Request Grouping
	6:0	**High Watermark** Default Value: 0100100b Access: R/W This is the number of write requests to be collected before initiating a write burst. Once a burst is initiated, it continues until all the available writes are requested.

ZTLB LRA 0

ZTLB_LRA_0 - ZTLB LRA 0	
Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x01007E00
Size (in bits):	32
Address:	04A30h

DWord	Bit	Description
0	31:29	**Reserved** Default Value: 000b Project: CHV, BSW Access: RO
	28:27	**STC LRA** Default Value: 00b Access: R/W Which LRA should STC use.
	26:18	**ZTLB LRA1 Min** Default Value: 001000000b Access: R/W Minimum value of programmable LRA1.
	17:9	**ZTLB LRA0 Max** Default Value: 000111111b Access: R/W Maximum value of programmable LRA0.
	8:0	**ZTLB LRA0 Min** Default Value: 000000000b Access: R/W Minimum value of programmable LRA0.

ZTLB LRA 1

ZTLB_LRA_1 - ZTLB LRA 1

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x33FD80BF CHV, BSW
Size (in bits):	32
Address:	04A34h

DWord	Bit	Description
0	31	**Reserved** Default Value: 0b Access: RO
	30:29	**HIZ LRA** Default Value: 01b Access: R/W Which LRA should HIZ use.
	28:27	**RCZ LRA** Default Value: 10b Access: R/W Which LRA should RCZ use.
	26:18	**ZTLB LRA2 Max** Access: R/W Maximum value of programmable LRA2. Value: 011111111b — Name: **[Default]** — Project: CHV, BSW
	17:9	**ZTLB LRA2 Min** Default Value: 011000000b Access: R/W Minimum value of programmable LRA2.
	8:0	**ZTLB LRA1 Max** Default Value: 010111111b Access: R/W Maximum value of programmable LRA1.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 2d: Command Reference: Structures

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

3DSTATE_CONSTANT(Body) 1
A32 Buffer Base Address Message Header Control 4
A64 Data Size Message Descriptor Control Field 5
A64 Dual Oword Block Message Header 6
A64 Hword Block Message Header 7
A64 Hword Data Blocks Message Descriptor Control Field 8
A64 Oword Block Message Header 9
A64 Oword Data Blocks Message Descriptor Control Field 10
A64 Oword Dual Data Blocks Message Descriptor Control Field 11
AddrSubRegNum 12
Any Binding Table Index Message Descriptor Control Field 13
Atomic Integer Binary Operation Message Descriptor Control Field 14
Atomic Integer Trinary Operation Message Descriptor Control Field 15
Atomic Integer Unary Operation Message Descriptor Control Field 16
AVC CABAC 17
AVC CAVLC 19
AVS_Inline_DMEM 21
BCS Hardware-Detected Error Bit Definitions 24
BINDING_TABLE_EDIT_ENTRY 25
BINDING_TABLE_STATE 26
Bit Definition for Interrupt Control Registers - Render 27
BLEND_STATE 29
BLEND_STATE_ENTRY 31
Block Dimensions Message Header Control 35
Block Message Header 36
BR00 - BLT Opcode and Control 37
BR01 - Setup BLT Raster OP, Control, and Destination Offset 41
BR05 - Setup Expansion Background Color 44
BR06 - Setup Expansion Foreground Color 45
BR07 - Setup Blit Color Pattern Address Lower Order Address bits 46
BR09 - Destination Address Lower Order Address Bits 47

BR11 - BLT Source Pitch (Offset) 48
BR12 - Source Address Lower order Address bits 49
BR13 - BLT Raster OP, Control, and Destination Pitch 50
BR14 - Destination Width and Height 53
BR15 - Color Pattern Address Lower order Address bits 54
BR16 - Pattern Expansion Background and Solid Pattern Color 55
BR17 - Pattern Expansion Foreground Color 56
BR18 - Source Expansion Background and Destination Color 57
BR19 - Source Expansion Foreground Color 58
BR27 - Destination Address Higher Order Address 59
BR28 - Source Address Higher order Address 60
BR29 - Color Pattern Address Higher order Address 61
BR30 - Setup Blit Color Pattern Address Higher Order Address 62
Byte Masked Media Block Message Header 63
Byte Masked Media Block Message Header Control 65
CC_VIEWPORT 67
Channel Mask Message Descriptor Control Field 68
Channel Mode Message Descriptor Control Field 69
COLOR_CALC_STATE 70
COLOR_PROCESSING_STATE - ACE State 72
COLOR_PROCESSING_STATE - CSC State 78
COLOR_PROCESSING_STATE - PROCAMP State 82
COLOR_PROCESSING_STATE - STD/STE State 83
COLOR_PROCESSING_STATE - TCC State 95
Color Calculator State Pointer Message Header Control 100
Color Code Message Header Control 101
Context Descriptor Format 102
Context Status 106
Data Port 0 Message Types 108
Data Port 1 Message Types 109
Data Size Message Descriptor Control Field 111
DstRegNum 112
DstSubRegNum 113

Dword Data Payload Register 114
Dword SIMD4x2 Atomic CMPWR Message Data Payload 116
Dword SIMD4x2 Atomic Operation Message Data Payload 117
Dword SIMD4x2 Data Payload 118
Dword SIMD8 Atomic Operation CMPWR Message Data Payload 120
Dword SIMD8 Data Payload 121
Dword SIMD16 Atomic Operation CMPWR Message Data Payload 122
Dword SIMD16 Data Payload 123
DX9_CONSTANTB_ENTRY 124
DX9_CONSTANTF_ENTRY 125
DX9_CONSTANTI_ENTRY 126
Encoder Statistics Format 127
EU_INSTRUCTION_BASIC_ONE_SRC 131
EU_INSTRUCTION_BASIC_THREE_SRC 132
EU_INSTRUCTION_BASIC_TWO_SRC 135
EU_INSTRUCTION_BRANCH_CONDITIONAL 136
EU_INSTRUCTION_BRANCH_ONE_SRC 138
EU_INSTRUCTION_BRANCH_TWO_SRC 139
EU_INSTRUCTION_COMPACT_THREE_SRC 140
EU_INSTRUCTION_COMPACT_TWO_SRC 143
EU_INSTRUCTION_CONTROLS_A 148
EU_INSTRUCTION_CONTROLS_B 151
EU_INSTRUCTION_CONTROLS 153
EU_INSTRUCTION_HEADER 154
EU_INSTRUCTION_ILLEGAL 155
EU_INSTRUCTION_MATH 156
EU_INSTRUCTION_NOP 157
EU_INSTRUCTION_OPERAND_CONTROLS 158
EU_INSTRUCTION_OPERAND_DST_ALIGN1 160
EU_INSTRUCTION_OPERAND_DST_ALIGN16 162
EU_INSTRUCTION_OPERAND_SEND_MSG 164
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 165
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 166

EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC....168
EU_INSTRUCTION_SEND....169
EU_INSTRUCTION_SOURCES_IMM32....171
EU_INSTRUCTION_SOURCES_REG....172
EU_INSTRUCTION_SOURCES_REG_IMM....173
EU_INSTRUCTION_SOURCES_REG_REG....174
ExtMsgDescpt....176
ExtMsgDescptImmediate....178
FFTID Message Header Control....180
Filter_Coefficient....181
Filter_Coefficients....182
FrameDeltaQp....183
FrameDeltaQpRange....184
FunctionControl....185
GATHER_CONSTANT_ENTRY....186
Hardware-Detected Error Bit Definitions....187
Hardware Status Page Layout....188
Header Forbidden Message Descriptor Control Field....193
Header Present Message Descriptor Control Field....194
Header Required Message Descriptor Control Field....195
HEVC_ARBITRATION_PRIORITY....196
HW Generated BINDING_TABLE_STATE....197
Hword 1 Block Data Payload....198
Hword 2 Block Data Payload....199
Hword 4 Block Data Payload....200
Hword 8 Block Data Payload....201
Hword Channel Mode Message Header Control....203
Hword Register Blocks Message Descriptor Control Field....204
Ignored Message Header....205
Inline Data Description for MFD_AVC_BSD_Object....206
Inline Data Description - VP8 PAK OBJECT....214
INTERFACE_DESCRIPTOR_DATA....218
Invalidate After Read Message Descriptor Control Field....224

JPEG 225
LOD Message Address Payload Control 226
Lower Oword Block Data Payload 227
MEDIA_SURFACE_STATE 228
MEMORY_OBJECT_CONTROL_STATE 235
MemoryAddressAttributes 236
Merged Media Block Message Header 238
Merged Media Block Message Header Control 240
Message Descriptor - Render Target Write 243
Message Descriptor - Sampling Engine 245
MFD_MPEG2_BSD_OBJECT Inline Data Description 247
MPEG2 250
MPEG4-2_Inline_DMEM 251
MsgDescpt31 257
Normal Media Block Message Header 258
Normal Media Block Message Header Control 260
oMask Message Data Payload Register 262
OM Replicated SIMD16 Render Target Data Payload 265
OM S0A SIMD8 Render Target Data Payload 266
OM S0A SIMD16 Render Target Data Payload 268
OM SIMD8 Dual Source Render Target Data Payload 270
OM SIMD8 Render Target Data Payload 272
OM SIMD16 Render Target Data Payload 273
Oword 1 Dual Block Data Payload 275
Oword 2 Block Data Payload 276
Oword 4 Block Data Payload 277
Oword 4 Dual Block Data Payload 278
Oword 8 Block Data Payload 280
Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload 281
Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload 282
Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload 283
Oword Data Blocks Message Descriptor Control Field 285
Oword Data Payload Register 286

Oword Dual Data Blocks Message Descriptor Control Field 287
PALETTE_ENTRY 288
Per Thread Scratch Space Message Header Control 289
Pixel Masked Media Block Message Header 290
Pixel Masked Media Block Message Header Control 292
Pixel Sample Mask Message Header Control 294
Pixel Sample Mask Render Target Message Header Control 295
Power Clock State Format 296
PPHWSP_LAYOUT 298
Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload 299
Qword Data Payload Register 301
Qword SIMD4x2 Atomic CMPWR8B Message Data Payload 302
Qword SIMD4x2 Atomic Operation Message Data Payload 303
Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload 304
Qword SIMD8 Atomic Operation CMPWR Message Data Payload 305
Qword SIMD8 Atomic Operation Return Data Message Data Payload 306
Qword SIMD8 Data Payload 307
Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload 308
Qword SIMD16 Atomic Operation Return Data Message Data Payload 310
Qword SIMD16 Data Payload 311
Read-Only Data Port Message Types 312
Read Surface Info 32-Bit Address Payload 313
Read Surface Info Data Payload 314
RENDER_SURFACE_STATE 316
Render Data Port Message Types 342
Render Target Index Message Header Control 343
Render Target Message Header 344
Render Target Message Header Control 346
Replicated Pixel Render Target Data Payload Register 349
Replicated SIMD16 Render Target Data Payload 350
Reversed SIMD Mode 2 Message Descriptor Control Field 351
RoundingPrecisionTable_3_Bits 352
S0A SIMD8 Render Target Data Payload 353

S0A SIMD16 Render Target Data Payload 354
SAMPLER_BORDER_COLOR_STATE 356
SAMPLER_INDIRECT_STATE_BORDER_COLOR 358
SAMPLER_INDIRECT_STATE 361
SAMPLER_STATE_8x8_AVS_COEFFICIENTS 364
SAMPLER_STATE_8x8_AVS 368
SAMPLER_STATE_8x8_CONVOLVE 380
SAMPLER_STATE_8x8_ERODE_DILATE_MINMAXFILTER 385
SAMPLER_STATE 386
SCISSOR_RECT 398
Scratch Hword Block Message Header 400
SF_CLIP_VIEWPORT 401
SF_OUTPUT_ATTRIBUTE_DETAIL 404
SFC_8x8_AVS_COEFFICIENTS 406
SIMD4x2 Typed Surface 32-Bit Address Payload 409
SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload 411
SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload 412
SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload 413
SIMD4x2 32-Bit Address Payload 414
SIMD8 Dual Source Render Target Data Payload 415
SIMD8 LOD Message Address Payload Control 417
SIMD8 Render Target Data Payload 419
SIMD8 Typed Surface 32-Bit Address Payload 420
SIMD8 Untyped BUFFER Surface 32-Bit Address Payload 421
SIMD8 Untyped BUFFER Surface 64-Bit Address Payload 422
SIMD8 Untyped STRBUF Surface 32-Bit Address Payload 423
SIMD16 Render Target Data Payload 424
SIMD16 Untyped BUFFER Surface 32-Bit Address Payload 426
SIMD16 Untyped BUFFER Surface 64-Bit Address Payload 427
SIMD16 Untyped STRBUF Surface 32-Bit Address Payload 428
SIMD 32-Bit Address Payload Control 429
SIMD 64-Bit Address Payload Control 431
SIMD8 32-Bit Address Payload 432

SIMD8 64-Bit Address Payload....433
SIMD16 32-Bit Address Payload....434
SIMD16 64-Bit Address Payload....435
SIMD Mode 2 Message Descriptor Control Field....436
SIMD Mode 3 Message Descriptor Control Field....437
SLM Surface Pixel Mask Message Header....438
Slot Group 2 Message Descriptor Control Field....439
Slot Group 3 Message Descriptor Control Field....440
Slot Group Select Render Cache Message Descriptor Control Field....441
SO_DECL....442
SO_DECL_ENTRY....444
SplitBaseAddress4KByteAligned....445
SplitBaseAddress64ByteAligned....446
SrcRegNum....447
SrcSubRegNum....448
Stateless Binding Table Index Message Descriptor Control Field....449
Stateless Block Message Header....450
Stateless Surface Message Header....452
Stateless Surface Pixel Mask Message Header....453
Subset Atomic Integer Trinary Operation Message Descriptor Control Field....454
Subset Reversed SIMD Mode 2 Message Descriptor Control Field....455
Subset SIMD Mode 2 Message Descriptor Control Field....456
Subset SIMD Mode 3 Message Descriptor Control Field....457
Subspan Render Target Message Header Control....458
Surface Binding Table Index Message Descriptor Control Field....459
Surface or Stateless Binding Table Index Message Descriptor Control Field....460
Surface Pixel Mask Message Header....461
SW Generated BINDING_TABLE_STATE....462
SZ OM S0A SIMD8 Render Target Data Payload....463
SZ OM S0A SIMD16 Render Target Data Payload....465
SZ OM SIMD8 Dual Source Render Target Data Payload....467
SZ OM SIMD8 Render Target Data Payload....469
SZ OM SIMD16 Render Target Data Payload....471

SZ S0A SIMD8 Render Target Data Payload 473
SZ S0A SIMD16 Render Target Data Payload 475
SZ SIMD8 Dual Source Render Target Data Payload 478
SZ SIMD8 Render Target Data Payload 480
SZ SIMD16 Render Target Data Payload 481
Thread Spawn Message Descriptor 483
TileW SIMD8 Data Control Dword 485
TileW SIMD8 Data Payload 486
Transpose Message Header 488
Untyped Write Channel Mask Message Descriptor Control Field 489
Upper Oword Block Data Payload 490
VC1 491
VCS Hardware-Detected Error Bit Definitions 492
VEBOX_CAPTURE_PIPE_STATE 493
VEBOX_Ch_Dir_Filter_Coefficient 496
VEBOX_DNDI_STATE 498
VEBOX_Filter_Coefficient 506
VEBOX_FORWARD_GAMMA_CORRECTION_STATE 507
VEBOX_GAMUT_STATE 512
VEBOX_RGB_TO_GAMMA_CORRECTION 529
VEBOX_STD_STE_STATE 530
VEBOX TLB Control Register 545
VEBOX_VERTEX_TABLE 546
VECS Hardware-Detected Error Bit Definitions 549
VERTEX_BUFFER_STATE 550
VERTEX_ELEMENT_STATE 553
Vertical Line Stride Override Message Descriptor Control Field 557
VFE_STATE_EX 558
VP8 Encoder StreamOut Format 561

3DSTATE_CONSTANT(Body)

3DSTATE_CONSTANT(Body)

Project: All

Source: RenderCS

Size (in bits): 320

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**Constant Buffer 1 Read Length** Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** (Project) • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 1. • If disabled, the **Pointer to Constant Buffer 1** must be programmed to zero. if gather constant are enabled, this field must be non-zero if a there was a preceding corresponding 3DSTATE_GATHER_CONSTANT_*., otherwise this field must be zero. (CHV, BSW)
	15:0	**Constant Buffer 0 Read Length** Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 0. • If disabled, the **Pointer to Constant Buffer 0** must be programmed to zero.
1	31:16	**Constant Buffer 3 Read Length** Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 3.

3DSTATE_CONSTANT(Body)

		• If disabled, the **Pointer to Constant Buffer 3** must be programmed to zero.
	15:0	**Constant Buffer 2 Read Length** Project: All Format: U16 read length This field specifies the length of the constant data to be loaded from memory in 256-bit units. **Programming Notes** • The sum of all four read length fields must be less than or equal to the size of 64 • Setting the value of the register to zero will disable buffer 2. • If disabled, the **Pointer to Constant Buffer 2** must be programmed to zero.
2..3 **Project:** CHV, BSW	63:5	**Pointer To Constant Buffer 0** Project: CHV, BSW Format: GraphicsAddress[63:5]ConstantBuffer **Description** / **Project**: When CONSTANT_BUFFER Address Offset Disable in INSTPM register is set, the value of this field is the virtual address of the location of the push constant buffer. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. When CONSTANT_BUFFER Address Offset Disable in INSTPM register is cleared, the value of this field is the offset into the Dynamic State Base Address. Only [47:5] of the field are added to the base address to generate the virtual address to be fetched from memory. / CHV, BSW
	4:0	**Reserved** Project: CHV, BSW Format: MBZ
4..5 **Project:** CHV, BSW	63:5	**Pointer To Constant Buffer 1** Project: CHV, BSW Format: GraphicsAddress[63:5]ConstantBuffer This field points to the location of Constant Buffer 1. If gather constants are enabled This field is an offset of constant Buffer1 from the Gather Pool BASE ADDRESS. If gather constants is disabled, the value of this field is the virtual address of the location of the push constant buffer. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Reserved**

3DSTATE_CONSTANT(Body)

		Project: All Format: MBZ
6..7 **Project:** CHV, BSW	63:5	**Pointer To Constant Buffer 2** Project: CHV, BSW Format: GraphicsAddress[63:5]ConstantBuffer The value of this field is the virtual address of the location of the push constant buffer 2. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Reserved** Project: CHV, BSW Format: MBZ
8..9 **Project:** CHV, BSW	63:5	**Pointer To Constant Buffer 3** Project: CHV, BSW Format: GraphicsAddress[63:5]ConstantBuffer The value of this field is the virtual address of the location of the push constant buffer 3. GraphicsAddress [63:48] are ignored by the HW and assumed to be in correct canonical form [63:48] == [47]. **Programming Notes** Constant buffers must be allocated in linear (not tiled) graphics memory.
	4:0	**Reserved** Project: CHV, BSW Format: MBZ

A32 Buffer Base Address Message Header Control

MHC_A32_BBA - A32 Buffer Base Address Message Header Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:0	**Buffer Base Address Offset** Project: All Format: GeneralStateOffset[31:0] Specifies the base address offset page [31:10] for A32 stateless messages.

A64 Data Size Message Descriptor Control Field

MDC_A64_DS - A64 Data Size Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**Data Size** Project: All Format: Enumeration Specifies the number of data elements to be read or written

Value	Name	Description	Project
00h	DE1	1 data element (B, DW, QW)	All
01h	DE2	2 data elements (B, DW, QW)	All
02h	DE4	4 data elements (B, DW, QW)	All
03h	DE8	8 data elements (B, DW, QW)	All

Restriction
The number of elements is constrained by SIMD Mode and Data Width. The max data payload limit is 256B: 2 elements SIMD16 QW, 4 elements SIMD16 DW, or 4 elements SIMD8 QW.

A64 Dual Oword Block Message Header

MH_A64_OWDB - A64 Dual Oword Block Message Header

Project:	CHV, BSW
Source:	DataPort 1
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**BlockOffset0** Project: All Format: U64 Specifies the U64 byte offset of Oword Block 0. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2-3	63:0	**BlockOffset1** Project: All Format: U64 Specifies the U64 byte offset of Oword Block 1. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
4-7	127:0	**Reserved** Project: All Format: Ignore Ignored

A64 Hword Block Message Header

MH_A64_HWB - A64 Hword Block Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**BlockOffset** Format: U64 Specifies the U64 byte offset of Oword block. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2-4	95:0	**Reserved** Format: Ignore Ignored
5	31:0	**Hword Channel Mode** Project: CHV, BSW Format: MHC_A64_CMODE [CHV, BSW] Specifies the Hword Channel Mode
6-7	63:0	**Reserved** Format: Ignore Ignored

A64 Hword Data Blocks Message Descriptor Control Field

MDC_A64_DB_HW - A64 Hword Data Blocks Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 3
Default Value: 0x00000001

DWord	Bit	Description
0	2:0	**Data Blocks** Project: All Format: Enumeration Specifies the number of Hwords to be read or written

Value	Name	Description	Project
01h	HW1 **[Default]**	1 Hword block	All
02h	HW2	2 Hword blocks	All
03h	HW4	4 Hword blocks	All
04h	HW8	8 Hword blocks	All
Others	Reserved	Ignored	All

A64 Oword Block Message Header

MH_A64_OWB - A64 Oword Block Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**BlockOffset** Project: All Format: U64 Specifies the U64 byte offset of Oword block. **Programming Notes** If the BlockOffset is not in the 48-bit canonical address range, the access is Out-of-Bounds. **Restriction** The byte offset must be aligned to the message's data type. Dwords have [1:0] = 0, Qwords have [2:0] = 0, and Hwords have [4:0] = 0.
2-7	191:0	**Reserved** Project: All Format: Ignore Ignored

A64 Oword Data Blocks Message Descriptor Control Field

MDC_A64_DB_OW - A64 Oword Data Blocks Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	3
Default Value:	0x00000000

DWord	Bit	Description
0	2:0	**Data Blocks** Project: All Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description	Project
00h	OW1L	1 Oword, read into or written from the low 128 bits of the destination register	All
01h	OW1U	1 Oword, read into or written from the high 128 bits of the destination register	All
02h	OW2	2 Owords	All
03h	OW4	4 Owords	All
04h	OW8	8 Owords	All
Others	Reserved	Ignored	All

A64 Oword Dual Data Blocks Message Descriptor Control Field

MDC_A64_DB_OWD - A64 Oword Dual Data Blocks Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	3
Default Value:	0x00000001

DWord	Bit	Description
0	2:0	**Data Blocks** Project: All Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description	Project
01h	OWD1 **[Default]**	1 Hword register, 2 Owords	All
03h	OWD4	4 Hword registers, 8 Owords	All
Others	Reserved	Ignored	All

AddrSubRegNum

AddrSubRegNum

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	4
Default Value:	0x00000000

Address Subregister Number This field provides the subregister number for the address register. The address register contains 8 sub-registers. The size of each subregister is one word. The address register contains the register address of the operand, when the operand is in register-indirect addressing mode. This field applies to the destination operand and the source operands. It is ignored (or not present in the instruction word) for an immediate source operand. This field is present if the operand is in register-indirect addressing mode; it is not present if the operand is directly addressed. An address subregister used for indirect addressing is often called an index register.

DWord	Bit	Description
0	3:0	**Address Subregister Number** Project: CHV, BSW Value: 0-15 — Name: Address Subregister Number

Any Binding Table Index Message Descriptor Control Field

MDC_BTS_SLM_A32 - Any Binding Table Index Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Project: All Format: Enumeration Specifies the surface for the message, which can be Surface State Model, SLM or Stateless.

Value	Name	Description	Project
00h-0EFh	BTS	Index of Binding Table State Surfaces	All
F0h-0FBh	Reserved	Reserved for future use	All
0FCh	Reserved	Reserved for future use	CHV, BSW
0FEh	SLM	Specifies an SLM access	All
0FFh	A32_A64	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)	All
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).	All

Restriction
When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Atomic Integer Binary Operation Message Descriptor Control Field

MDC_AOP2 - Atomic Integer Binary Operation Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x00000001

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Project: All Format: Enumeration Specifies the atomic integer binary operation to be performed

Value	Name	Description	Project
01h	AOP_AND **[Default]**	new_dst = old_dst AND src0	All
02h	AOP_OR	new_dst = old_dst | src0	All
03h	AOP_XOR	new_dst = old_dst ^ src0	All
04h	AOP_MOV	new_dst = src0	All
07h	AOP_ADD	new_dst = old_dst + src0	All
08h	AOP_SUB	new_dst = old_dst - src0	All
09h	AOP_REVSUB	new_dst = src0 - old_dst	All
0Ah	AOP_IMAX	new_dst = imax(old_dst, src0)	All
0Bh	AOP_IMIN	new_dst = imin(old_dst, src0)	All
0Ch	AOP_UMAX	new_dst = umax(old_dst, src0)	All
0Dh	AOP_UMIN	new_dst = umin(old_dst, src0)	All
Others	Reserved	Ignored	All

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Integer Trinary Operation Message Descriptor Control Field

MDC_AOP3 - Atomic Integer Trinary Operation Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x0000000E

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Project: All Format: Enumeration Specifies the atomic integer trinary operation to be performed

Value	Name	Description	Project
00h	AOP_CMPWR_2W	new_dst = (src0_2W == old_dst_2W) ? src1_2W : old_dst_2W	All
0Eh	AOP_CMPWR **[Default]**	new_dst = (src0 == old_dst) ? src1 : old_dst	All
Others	Reserved	Ignored	All

Programming Notes
When Return Data Control is set, old_dst is returned.

Atomic Integer Unary Operation Message Descriptor Control Field

MDC_AOP1 - Atomic Integer Unary Operation Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x00000005

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Project: All Format: Enumeration Specifies the atomic integer unary operation to be performed

Value	Name	Description	Project
05h	AOP_INC **[Default]**	new_dst = old_dst + 1	All
06h	AOP_DEC	new_dst = old_dst - 1	All
0Fh	AOP_PREDEC	new_dst = old_dst - 1	All
Others	Reserved	Ignored	All

Programming Notes
When Return Data Control is set, new_dst is returned by AOP_PREDEC and otherwise old_dst is returned.

AVC CABAC

AVC CABAC		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	

DWord	Bit	Description
0	15	**Reserved** Format: MBZ
	14	**Coefficient level out-of-bound Error** This flag indicates the coded coefficient level SEs in the bit-stream is out-of-bound.
	13	**Reserved** Format: MBZ
	12	**Reserved** Format: MBZ
	11	**Temporal Direction Motion Vector Out-of-Bound Error** This flag indicates motion vectors calculated from Temporal Direct Motion Vector is larger than the allowed range specified by the AVC spec.
	10	**Reserved** MBZ
	9	**Motion Vector Delta SE Out-of-Bound Error** This flag indicates inconsistent Motion Vector Delta SEs coded in the bit-stream.
	8	**Reference Index SE Out-of-Bound Error** This flag indicates inconsistent Reference Index SEs coded in the bit-stream.
	7	**MacroBlock QpDelta Error** This flag indicates out-of-bound MB QP delta SEs coded in the bit-stream.
	6	**Motion Vector Delta SE Error** This flag indicates out-of-bound motion vector delta SEs coded in the bit-stream.
	5	**Reference Index SE Error** This flag indicates out-of-bound Refidx SEs coded in the bit-stream.
	4	**Residual Error** This flag indicates out-of-bound absolute coefficient level SEs coded in the bit-stream.
	3	**Slice end Error** This flag indicates a pre-matured slice_end SE or inconsistent slice end on the last MB of a slice.
	2	**Chroma Intra prediction Mode Error** This flag indicates inconsistent Chroma Intra prediction mode SEs coded in the bit-stream.
	1	**Luma Intra prediction Mode Error** This flag indicates inconsistent luma Intra prediction mode SE coded in the bit-stream.

AVC CABAC		
	0	**MB Concealment Flag** Each pulse from this flag indicates one MB is concealed by hardware.

AVC CAVLC

AVC CAVLC		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Total Zero out-of-bound Error** This flag indicates the Total zero SE count exceed the max number of coeffs allowed in an intra16x16 AC block.
	14	**Coefficient level out-of-bound Error** This flag indicates the coded coefficient level SEs in the bit-stream is out-of-bound.
	13	**RunBefore out-of-bound Error** This flag indicates the coded RunBefore SE value is larger than the remaining zero block count.
	12	**Total coefficient Out-of-bound Error** This flag indicates the coded total coeff SE count exceed the max number of coeffs allowed in an intra16x16 AC block.
	11	**Temporal Direction Motion Vector Out-of-Bound Error** This flag indicates motion vectors calculated from Temporal Direct Motion Vector is larger than the allowed range specified by the AVC spec.
	10	**Reserved** Reserved
	9	**Motion Vector Delta SE Out-of-Bound Error** This flag indicates inconsistent Motion Vector Delta SEs coded in the bit-stream.
	8	**Reference Index SE Out-of-Bound Error** This flag indicates inconsistent Reference Index SEs coded in the bit-stream.
	7	**RunBefore/TotalZero Error** This flag indicates one or more inconsistent RunBefore or TotalZero SEs coded in the bit-stream.
	6	**Exponential Golomb Error** This flag indicates hardware detects more than 18 leadzero for skip and more than 19 for other SEs from the Exponential Golomb Logic
	5	**Total Coeff SE Error** This flag indicates one or more inconsistent total coeff SEs coded in the bit-stream.
	4	**Macroblock Coded Block Pattern Error** This flag indicates inconsistent CBP SEs coded in the bit-stream.
	3	**Mbytpe/submbtype Error** This flag indicates inconsistent MBtype/SubMBtype SEs coded in the bit-stream.
	2	**Chroma Intra prediction Mode Error** This flag indicates inconsistent Chroma Intra prediction mode SEs coded in the bit-stream.
	1	**Luma Intra prediction Mode Error**

AVC CAVLC		
		This flag indicates inconsistent luma Intra prediction mode SE coded in the bit-stream.
	0	**MB Concealment Flag** Each pulse from this flag indicates one MB is concealed by hardware.

AVS_Inline_DMEM

AVS_Inline_DMEM		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	768	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:3	**Reserved** Format: MBZ
	2:0	**hw_used_bits** Index into the first valid bit of the starting byte of the first macroblock of the frame
1	31:14	**Reserved** Format: MBZ
	13:0	**hw_width** AVS SE: horizontal_size
2	31:14	**Reserved** Format: MBZ
	13:0	**hw_height** AVS SE: vertical_size
3	31:0	**hw_MbMax** In the VideoSequenceHeader of the AVS standard specification: MbWidth*MbHeight
4	31:2	**Reserved** Format: MBZ
	1:0	**hw_chroma_format** AVS SE: chroma_format
5	31:2	**Reserved** Format: MBZ
	1:0	**hw_picture_coding_type** AVS SE: picture_coding_type
6	31:8	**Reserved** Format: MBZ
	7:0	**hw_picture_distance** AVS SE: picture_distance
7	31:1	**Reserved** Format: MBZ

AVS_Inline_DMEM		
	0	**hw_picture_structure** AVS SE: picture_structure
8	31:0	**Reserved** Format: MBZ
9	31:0	**Reserved** Format: MBZ
10	31:0	**Reserved** Format: MBZ
11	31:0	**Reserved** Format: MBZ
12	31:0	**Reserved** Format: MBZ
13	31:0	**Reserved** Format: MBZ
14	31:0	**Reserved** Format: MBZ
15	31:0	**Reserved** Format: MBZ
16	31:0	**Reserved** Format: MBZ
17	31:0	**Reserved** Format: MBZ
18	31:1	**Reserved** Format: MBZ
	0	**hw_fixed_picture_qp** AVS SE: fixed_picture_qp
19	31:6	**Reserved** Format: MBZ
	5:0	**hw_picture_qp** AVS SE: picture_qp
20	31:1	**Reserved** Format: MBZ
	0	**hw_picture_reference_flag** AVS SE: picture_reference_flag
21	31:1	**Reserved** Format: MBZ

AVS_Inline_DMEM		
	0	**hw_skip_mode_flag** AVS SE: skip_mode_flag
22	31:1	**Reserved** Format: MBZ
	0	**hw_loop_filter_diable** AVS SE: loop_filter_disable
23	31:0	**Reserved** Format: MBZ

BCS Hardware-Detected Error Bit Definitions

BCS Hardware-Detected Error Bit Definitions

Project: CHV, BSW
Source: BlitterCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** Project: CHV, BSW This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: Value: 1 — Name: — Description: Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

BINDING_TABLE_EDIT_ENTRY

BINDING_TABLE_EDIT_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:24	**Reserved** Format: MBZ
	23:16	**Binding Table Index** Format: U8 This field specifies the index of binding table entry that will be updated.
	15:0	**Surface State Pointer** Format: SurfaceStateOffset[21:6]RENDER_SURFACE_STATE [CHV, BSW] Surface State Pointer. This address points to a surface state block. This pointer is relative to the Surface State Base Address.

BINDING_TABLE_STATE

BINDING_TABLE_STATE

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

The binding table binds surfaces to logical resource indices used by shaders and other compute engine kernels. It is stored as an array of up to 256 elements, each of which contains one dword as defined here. The start of each element is spaced one dword apart. The first element of the binding table is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31:6	**Surface State Pointer** Project: CHV, BSW Format: SurfaceStateOffset[31:6] This 64-byte aligned address points to a surface state block. This pointer is relative to the **Surface State Base Address**.
	5:0	**Reserved** Project: CHV, BSW Format: MBZ

Bit Definition for Interrupt Control Registers - Render

Bit Definition for Interrupt Control Registers - Render

Project:	CHV, BSW
Source:	RenderCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Project: CHV, BSW Format: MBZ Reserved for other command streamers - cannot be allocated by main command streamer.
	15:12	**Reserved** Project: CHV, BSW Format: MBZ
	11	**Wait on Semaphore** Project: CHV, BSW Exec-List Scheduling: Set when MI_SEMAPHORE_WAIT command is un-successful and when "Inhibit Synchronous Context Switch" is set. Scheduler can use this interrupt to preempt the context waiting on semaphore wait.
	10	**L3 Counter Save Interrupt** Project: CHV, BSW
	9	**Reserved** Project: CHV, BSW Format: MBZ
	8	**Context Switch Interrupt** Project: CHV, BSW Set when a context switch has just occurred. Execlist Enable bit needs to be set for this interrupt to occur.
	7	**Page Fault** Project: All Description: This interrupt is for handling Legacy Page Fault interface for all Command Streamers (BCS, RCS, VCS, VECS). When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c Project: CHV, BSW

Bit Definition for Interrupt Control Registers - Render

		"Page Fault Support" section for more details.
	6	**Timeout Counter Expired** Set when the render pipe timeout counter (0x02190) has reached the timeout threshold value (0x0217c).
	5	**L3 Parity Error (Slice0)** Project: CHV, BSW When this bit is set, L3 cache controller is indicating that it has encountered an parity error while checking the data.
	4	**PIPE_CONTROL Notify Interrupt** The Pipe Control packet (Fences) specified in 3D pipeline document may optionally generate an Interrupt. The Store QW associated with a fence is completed ahead of the interrupt.
	3	**Render Command Parser Master Error** When this status bit is set, it indicates that the hardware has detected an error. It is set by the device upon an error condition and cleared by a CPU write of a one to the appropriate bit contained in the Error ID register followed by a write of a one to this bit in the IIR. Further information on the source of the error comes from the "Error Status Register" which along with the "Error Mask Register" determine which error conditions will cause the error status bit to be set and the interrupt to occur. **Page Table Error:** Indicates a page table error. **Instruction Parser Error:** The Render Instruction Parser encounters an error while parsing an instruction.
	2	**Reserved** Project: CHV, BSW Format: MBZ
	1	**Reserved**
	0	**Render Command Parser User Interrupt** This status bit is set when an MI_USER_INTERRUPT instruction is executed on the Render Command Parser. Note that instruction execution is not halted and proceeds normally. A mechanism such as an MI_STORE_DATA instruction is required to associate a particular meaning to a user interrupt.

BLEND_STATE

BLEND_STATE

Project:	CHV, BSW
Source:	PRM
Size (in bits):	544
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

The blend state is stored as a structure containing a common DWORD that applies to all RTs and an array of up to 8 elements, each of which contains the two DWords for each. The start of each element is spaced 2 DWords apart. The blend state is aligned to a 64-byte boundary, which is pointed to by a field in 3DSTATE_BLEND_STATE_POINTERS. The 3-bit Render Target Index field in the Render Target Write data port message header is used to select which of the 8 elements from BLEND_STATE that is used on the current message.

DWord	Bit	Description
0	31	**Alpha To Coverage Enable** Format: Enable If set, Source0 Alpha is converted to a temporary 1/2/4-bit coverage mask and the mask bit corresponding to the sample# ANDed with the sample mask bit. If set, sample coverage is computed based on src0 alpha value. Value of 0 disables all samples and value of 1 enables all samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels.The field is applied to all the RTs in MRT case.
	30	**Independent Alpha Blend Enable** Format: Enable When enabled, the other fields in this instruction control the combination of the alpha components in the Color Buffer Blend stage. When disabled, the alpha components are combined in the same fashion as the color components.The field is applied to all the RTs in MRT case.
	29	**Alpha To One Enable** Format: Enable If set, Source0 Alpha is set to 1.0f after (possibly) being used to generate the AlphaToCoverage coverage mask.If Dual Source Blending is enabled, this bit must be disabled.The field is applied to all the RTs in MRT case.
	28	**Alpha To Coverage Dither Enable** Format: Enable If set, sample coverage is computed based on src0 alpha value and it modulates the sample coverage based on screen coordinates. Value of 0 disables all samples and value of 1 enables all

BLEND_STATE

		samples for that pixel. The same coverage needs to apply to all the RTs in MRT case. Further, any value of src0 alpha between 0 and 1 monotonically increases the number of enabled pixels. If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.The field is applied to all the RTs in MRT case.
	27	**Alpha Test Enable** Format: Enable Enables the AlphaTest function of the Pixel Processing pipeline.The field is applied to all the RTs in MRT case. **Programming Notes** Alpha Test can only be enabled if Pixel Shader outputs a float alpha value. Alpha Test is applied independently on each render target by comparing that render target's alpha value against the alpha reference value. If the alpha test fails, the corresponding pixel write will be supressed only for that render target. The depth/stencil update will occur if alpha test passes for any render target.
	26:24	**Alpha Test Function** Format: 3D_Compare_Function This field specifies the comparison function used in the AlphaTest functionThe field is applied to all the RTs in MRT case.
	23	**Color Dither Enable** Format: Enable Enables dithering of colors (including any alpha component) before they are written to the Color Buffer. The field is applied to all the RTs in MRT case. **Programming Notes** For YUV render target formats, this field must be programmed to 0.
	22:21	**X Dither Offset** Format: U2 Specifies offset to apply to pixel X coordinate LSBs when accessing dither table.The field is applied to all the RTs in MRT case.
	20:19	**Y Dither Offset** Format: U2 Specifies offset to apply to pixel Y coordinate LSBs when accessing dither table.The field is applied to all the RTs in MRT case.
	18:0	**Reserved** Format: MBZ
1..16	63:0	**Entry** Format: BLEND_STATE_ENTRY [CHV, BSW]

BLEND_STATE_ENTRY

BLEND_STATE_ENTRY

Project: CHV, BSW
Source: PRM
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0	63	**Logic Op Enable** Format: Enable Enables the LogicOp function of the Pixel Processing pipeline. **Programming Notes** Enabling LogicOp and Color Buffer Blending at the same time is UNDEFINED
	62:59	**Logic Op Function** Format: 3D_Logic_Op_Function This field specifies the function to be performed (when enabled) in the Logic Op stage of the Pixel Processing pipeline. Note that the encoding of this field is one less than the corresponding "R2_" ROP code defined in WINGDI.H, and is a rather contorted mapping of the OpenGL LogicOp encodings. However, this field was defined such that, when the 4 bits are replicated to 8 bits, they coincide with the ROP codes used in the Blter. Note: if the Logic Op Function does not depend on "D", the dest buffer is not read.
	58:37	**Reserved** Format: MBZ
	36	**Pre-Blend Source Only Clamp Enable** This field specifies whether the source(s) are clamped prior to blending, regardless of whether blending is enabled. If DISABLED, no clamping is performed prior to blending. If ENABLED, only source0 and source 1, if dual source is enabled, are clamped prior to the blend to the range specified by Color Clamp Range. Value / Name / Description 0 / Disabled / No clamping is performed prior to blending. 1 / Enabled / Only Source(s) are clamped prior to blend function. Other inputs to blend must not be clamped. **Programming Notes** See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. **When this bit is enabled Pre-Blend Color Clamp Enable RT[0] must be disabled.**
	35:34	**Color Clamp Range**

BLEND_STATE_ENTRY

Specifies the clamped range used in Pre-Blend and Post-Blend Color Clamp functions if one or both of those functions are enabled. Note that this range selection is shared between those functions. This field is ignored if both of the Color Clamp Enables are disabled

Value	Name	Description
0	COLORCLAMP_UNORM	Clamp Range [0,1]
1	COLORCLAMP_SNORM	Clamp Range [-1,1]
2	COLORCLAMP_RTFORMAT	Clamp to the range of the RT surface format (Note: The Alpha component is clamped to FLOAT16 for R11G11B10_FLOAT format).
3	Reserved	Reserved

33 **Pre-Blend Color Clamp Enable**

Format:	Enable

This field specifies whether the source, destination and constant color channels are clamped prior to blending, regardless of whether blending is enabled.If DISABLED, no clamping is performed prior to blending.If ENABLED, all inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.

Value	Name	Description
0	Disabled	No clamping is performed prior to blending.
1	Enabled	All inputs to the blend function are clamped prior to the blend to the range specified by Color Clamp Range.

Programming Notes
See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range.

32 **Post-Blend Color Clamp Enable**

Format:	Enable

If blending is enabled, this field specifies whether the blending output channels are first clamped to the range specified by Color Clamp Range. Regardless of whether this clamping is enabled, the blending output channels will be clamped to the RT surface format just prior to being written.

Programming Notes
See table in Pre-Blending Color Clamp subsection for programming restrictions as a function of RT format. This field is ignored (treated as DISABLED) for UINT and SINT RT surface formats. Blending is not supported for those RT surface formats. The device will automatically clamp source color channels to the respective RT surface range. **When this bit is enabled Pre-Blend Source Only Clamp Enable RT[0] must be disabled.**

31 **Color Buffer Blend Enable**

Format:	Enable

Enables the ColorBufferBlending (nee "alpha blending") function of the Pixel Processing Pipeline

BLEND_STATE_ENTRY

for this render target.

Programming Notes
Enabling LogicOp and ColorBufferBlending at the same time is UNDEFINED

30:26 Source Blend Factor

Format:	3D_Color_Buffer_Blend_Factor

Controls the "source factor" in the ColorBufferBlending function.Refer to Source Alpha Blend Factor for encodings.

25:21 Destination Blend Factor

Format:	3D_Color_Buffer_Blend_Factor

Controls the "destination factor" in the ColorBufferBlending function. Refer to Source Alpha Blend Factor for encodings.

20:18 Color Blend Function

Format:	3D_Color_Buffer_Blend_Function

This field specifies the function used to combine the color components in the ColorBufferBlending function of the Pixel Processing Pipeline. If Independent Alpha Blend Enable is disabled, this field will also control the blending of the alpha components in the ColorBufferBlending function.

17:13 Source Alpha Blend Factor

Format:	3D_Color_Buffer_Blend_Factor

Controls the "source factor" in alpha Color Buffer Blending stage.Note: For the source/destination alpha blend factors, the encodings indicating "COLOR" are the same as the encodings indicating "ALPHA", as the alpha component of the color is selected.

12:8 Destination Alpha Blend Factor

Format:	3D_Color_Buffer_Blend_Factor

Controls the "destination factor" in alpha Color Buffer Blending stage. Refer to Source Alpha Blend Factor for encodings.

7:5 Alpha Blend Function

Format:	3D_Color_Buffer_Blend_Function

This field specifies the function used to combine the alpha components in the Color Buffer blend stage of the Pixel Pipeline when the IndependentAlphaBlend state is enabled.

4 Reserved

Format:	MBZ

3 Write Disable Alpha

Format:	Disable

This field controls the writing of the alpha component into the Render Target.

BLEND_STATE_ENTRY

Value	Name	Description
0b	Enabled	Alpha component can be overwritten
1b	Disabled	Writes to the color buffer will not modify Alpha.

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

2 **Write Disable Red**

Format:	Disable

This field controls the writing of the red component into the Render Target.

Value	Name	Description
0b	Enabled	Red component can be overwritten
1b	Disabled	Writes to the color buffer will not modify Red.

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

1 **Write Disable Green**

Format:	Disable

This field controls the writing of the green component into the Render Target.

Value	Name	Description
0b	Enabled	Green component can be overwritten
1b	Disabled	Writes to the color buffer will not modify Green.

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

0 **Write Disable Blue**

Format:	Disable

This field controls the writing of the Blue component into the Render Target.

Value	Name	Description
0b	Enabled	Blue component can be overwritten
1b	Disabled	Writes to the color buffer will not modify Blue.

Programming Notes
For YUV surfaces, this field must be set to 0B (enabled).

Block Dimensions Message Header Control

MHC_BDIM - Block Dimensions Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:22	**Reserved** Project: All Format: Ignore Ignored
	21:20	**Block Height** Project: All Format: Enumeration Height in rows of block being accessed. Range = [0,3] representing 1 to 8 rows. Value / Name / Description / Project 0h / H1 / Block height = 1 row / All 1h / H2 / Block height = 2 rows / All 2h / H4 / Block height = 4 rows / All 03h / H8 / Block height = 8 rows / All
	19:2	**Reserved** Project: All Format: Ignore Ignored
	1:0	**Block Width** Project: All Format: Enumeration Width in Dwords of block being accessed. Range = [0,3] representing 1 to 8 Dwords. Value / Name / Description / Project 0h / W1 / Block width = 1 Dword / All 1h / W2 / Block width = 2 Dwords / All 2h / W4 / Block width = 4 Dwords / All 03h / W8 / Block width = 8 Dwords / All

Block Message Header

MH_BTS_GO - Block Message Header

Project: CHV, BSW

Source: DataPort 0

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**Reserved** Project: All Format: Ignore Ignored
2	31:0	**Global Offset** Project: All Format: U32 Specifies the global element index into the buffer, in units of Owords, Dwords, or Bytes (depending on the message). **Programming Notes** The Global Offset for Oword Unaligned Block operations is specified as a Dword-aligned byte offset (offset bits [1:0] = 0). If the address offset calculated with the Global Offset is greater than the Surface Size, then the access is Out-of-Bounds.
3-7	159:0	**Reserved** Project: All Format: Ignore Ignored

BR00 - BLT Opcode and Control

BR00 - BLT Opcode and Control

Project: CHV, BSW
Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**BLT Engine Busy** This bit indicates whether the BLT Engine is busy (1) or idle (0). This bit is replicated in the SETUP BLT Opcode and Control register. Value: 0 — Name: Idle **[Default]** Value: 1 — Name: Busy
	30	**Setup Instruction Instruction** Default Value: 0 The current instruction performs clipping (1).
	29	**Setup Monochrome Pattern** This bit is decoded from the Setup instruction opcode to identify whether a color (0) or monochrome (1) pattern is used with the SCANLINE_BLT instruction. Value: 0 — Name: Color **[Default]** Value: 1 — Name: Monochrome
	28:22	**Instruction Target (Opcode)** Default Value: 0000000b This is the contents of the Instruction Target field from the last BLT instruction. This field is used by the BLT Engine state machine to identify the BLT instruction it is to perform. The opcode specifies whether the source and pattern operands are color or monochrome.
	21:20	**32bpp Byte Mask** This field is only used for 32bpp. Value: 00b — Name: **[Default]** Value: 1xb — Name: Write Alpha Channel Value: x1b — Name: Write RGB Channel
	19:17	**Monochrome Source Start** Default Value: 000b This field indicates the starting monochrome pixel bit position within a byte per scan line of the

BR00 - BLT Opcode and Control

source operand. The monochrome source is word aligned which means that at the end of the scan line all bits should be discarded until the next word boundary.

16 **Bit/Byte Packed**

Byte packed is for the NT driver.

Value	Name
0b	Bit **[Default]**
1b	Byte

15 **Src Tiling Enable**

Value	Name	Project
0b	Tiling Disabled (Linear) **[Default]**	
1b	Tiling enabled: Tile-X or Tile-Y	CHV, BSW

14:12 **Horizontal Pattern Seed**

Default Value:	0b

This field indicates the pattern pixel position which corresponds to X = 0.

11 **Dest Tiling Enable**

When set to '1', this means that Blitter is executing in Tiled mode. If '0' it means that Blitter is in Linear mode. Pre-Dev Blitter never executes in Tiled-Y mode, DevGT+ Blitter supports both Tile-X and Tile-Y modes. On reset, this bit will be '0'. This definition applies to only X, Y Blits.

Value	Name	Project
0b	Tiling Disabled (Linear blit) **[Default]**	
1b	Tiling enabled: Tile-X or Tile-Y	CHV, BSW

10:8 **Transparency Range Mode**

These bits control whether or not the byte(s) at the destination corresponding to a given pixel will be conditionally written, and what those conditions are. This feature can make it possible to perform various masking functions in order to selectively write or preserve graphics data already at the destination.

Value	Name	Description
xx0b	**[Default]**	No color transparency mode enabled. This causes normal operation with regard to writing data to the destination.
001b		[Source color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (R, G, B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
011b		[Source and Alpha color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color

BR00 - BLT Opcode and Control

				High: (Pixel Less or Equal) (source foreground register) are compared to the source pixels. The range comparisons are done on each component (A, R, G, B) and then logically ANDed. If the source pixel components are not within the range defined by the Transparency Color registers, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation."
		101b		[Destination and Alpha color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (A, R, G, B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
		111b		[Destination color transparency] The Transparency Color Low: (Pixel Greater or Equal) (source background register) and the Transparency Color High: (Pixel Less or Equal) (source foreground register) are compared to the destination pixels. The range comparisons are done on each component (R, G, B) and then logically ANDed. If the destination pixels are within the range, then the byte(s) at the destination corresponding to the current pixel are written with the result of the bit-wise operation.
	7:5	**Pattern Vertical Seed**		
		Default Value:	000b	
		This field specifies the pattern scan line which corresponds to Y=0.		
	4	**Destination Read Modify Write**		
		Default Value:	0b	
		This bit is decoded from the last instruction's opcode field and Destination Transparency Mode to identify whether a Destination read is needed.		
	3	**Color Source**		
		Default Value:	0b	
		This bit is decoded from the last instructions opcode field to identify whether a color (1) source is used.		
	2	**Monochrome Source**		
		Default Value:	0b	
		This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) source is used.		
	1	**Color Pattern**		
		Default Value:	0b	
		This bit is decoded from the last instructions opcode field to identify whether a color (1) pattern		

BR00 - BLT Opcode and Control		
		is used.
	0	**Monochrome Pattern** Default Value: 0b This bit is decoded from the last instructions opcode field to identify whether a monochrome (1) pattern is used.

BR01 - Setup BLT Raster OP, Control, and Destination Offset

BR01 - Setup BLT Raster OP, Control, and Destination Offset

Project: CHV, BSW
Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Solid Pattern Select** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations.
1b		The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.

DWord	Bit	Description
	30	**Clipping Enabled**

Value	Name
0b	**[Default]**
1b	

DWord	Bit	Description
	29	**Monochrome Source Transparency Mode** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1b		Wherever a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the

BR01 - Setup BLT Raster OP, Control, and Destination Offset

		destination are allowed to remain unchanged.

28 Monochrome Pattern Transparency Mode

This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field.

Value	Name	Description
0b	**[Default]**	This causes normal operation with regard to the use of the pattern data. Wherever a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1b		Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

27:26 32bpp Byte Mask

This bit applies only when the pattern data is monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode field.

Value	Name
00b	**[Default]**
1xb	Write Alpha Channel
x1b	Write RGB Channel

25:24 Color Depth

Value	Name
00b	8 Bit Color Depth **[Default]**
01b	16 Bit Color Depth
10b	Alternate 16 Bit Color Depth
11b	32 Bit Color Depth

23:16 Raster Operation Select

These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine.

15:0 Destination Pitch (Offset)

For non-XY Blits, the signed 16bit field allows for specifying upto + 32Kbytes signed pitches in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Destination will be 512Byte aligned and should be programmable upto + 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Destination will be 128Byte aligned and should be programmable upto +

BR01 - Setup BLT Raster OP, Control, and Destination Offset		
		128Kbytes. In this case, this 16bit signed pitch field is used to specify upto + 32KDWords. For X, Y blits with nontiled surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto + 32Kbytes (same as before). These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

BR05 - Setup Expansion Background Color

BR05 - Setup Expansion Background Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Setup Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. BR05 is also used as the solid pattern for the PIXEL_BLT instruction. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR06 - Setup Expansion Foreground Color

BR06 - Setup Expansion Foreground Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Setup Expansion Foreground Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern or source data for either the SCANLINE_BLT or TEXT_BLT instructions. Whether one, two, or three bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR07 - Setup Blit Color Pattern Address Lower Order Address bits

BR07 - Setup Blit Color Pattern Address Lower Order Address bits

Project:	CHV, BSW
Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:6	**Setup Blit Color Pattern Address** Format: GraphicsAddress[31:6] Lower 32bits of the 48bit addressing. These 26 bits specify the starting address of the (8X8) pixel color **pattern from the SETUP_BLT instruction.** This register works identically to the Pattern Address register (BR15), but this version is **only used with the SCANLINE_BLT instruction execution** (the actual programming for this, is done in XY_SETUP_BLT command). The pattern data must be located in linear memory. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and is supplied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
	5:0	**Reserved** Format: MBZ

BR09 - Destination Address Lower Order Address Bits

BR09 - Destination Address Lower Order Address Bits		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Destination Address Bits** Format: GraphicsAddress[31:0] When tiling is enabled for XY-blits, this base address should be limited to 4KB. when tiling is disabled for XY-blits, this base address should be CL (64byte) aligned. These lower 32bits of the 48bit address, which specify the starting pixel address of the destination data. This register is also the working destination address register for the lower 32bits of the address, and changes as the BLT Engine performs the accesses. Used as the scan line address (Destination Y Address and Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 and Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text. Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), this address points to the first byte to be written. Note: Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

BR11 - BLT Source Pitch (Offset)

BR11 - BLT Source Pitch (Offset)		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:16	**Reserved**
	15:0	**Source Pitch (Offset)** For non-XY Blits with color source operand (SRC_COPY_BLT), the signed 16bit field allows for specifying upto + 32Kbytes signed pitch in bytes (same as before). For X, Y Blits with tiled-X surfaces, the pitch for Color Source will be 512Byte aligned and should be programmable upto + 128Kbytes. For X, Y Blits with tiled-Y surfaces, the pitch for Color Source will be 128Byte aligned and should be programmable upto + 128Kbytes. In this case, this 16bit signed pitch field is used to specify upto + 32KDWords. For X, Y blits with nontiled color source surfaces (linear surfaces), this 16bit field can be programmed to byte specification of upto + 32Kbytes (same as before). When the color source data is located within the frame buffer or AGP aperture, these signed 16 bits store the memory address offset (pitch) value by which the source address originally specified in the Source Address Register is incremented or decremented as each scan line's worth of source data is read from the frame buffer by the BLT Engine, so that the source address will point to the next memory address from which the next scan line's worth of source data is to be read. Note that if the intended source of a BLT operation is within on-screen frame buffer memory, this offset is normally set to accommodate the fact that each subsequent scan line's worth of source data lines up vertically with the source data in the scan line, above. However, if the intended source of a BLT operation is within off-screen memory, this offset can be set to accommodate a situation in which the source data exists as a single contiguous block of bytes where in each subsequent scan line's worth of source data is stored at a location immediately after the location where the source data for the last scan line ended.

BR12 - Source Address Lower order Address bits

BR12 - Source Address Lower order Address bits

Project:	CHV, BSW
Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:0	**Source Address Bits** Format: GraphicsAddress[31:0] Lower 32bits of the 48bit addressing. When tiling is enabled for XY-blits with Color source surfaces, this base address should be limited to 4KB. When tiling is disabled for XY-blits, this base address should be CL (64byte) aligned. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read. These lower 32bits of the 48bit address, specify the starting pixel address of the color source data. The lower 3 bits are used to indicate the position of the first valid byte within the first Quadword of the source data. If this Source happens to be a Monosource surface, then this Monosource Base Address programmed, must always be Cache Line (64byte) aligned.

BR13 - BLT Raster OP, Control, and Destination Pitch

BR13 - BLT Raster OP, Control, and Destination Pitch

Project: CHV, BSW
Source: BlitterCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Solid Pattern Select** This bit applies only when the pattern data is monochrome. This bit determines whether or not the BLT Engine actually performs read operations from the frame buffer in order to load the pattern data. Use of this feature to prevent these read operations can increase BLT Engine performance, if use of the pattern data is indeed not necessary. The BLT Engine is configured to accept either monochrome or color pattern data via the opcode field.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the pattern data. The BLT Engine proceeds with the process of reading the pattern data, and the pattern data is used as the pattern operand for all bit-wise operations.
1		The BLT Engine forgoes the process of reading the pattern data, the presumption is made that all of the bits of the pattern data are set to 0, and the pattern operand for all bit-wise operations is forced to the background color specified in the Color Expansion Background Color Register.

DWord	Bit	Description
	30	**Clipping Enabled**

Default Value:	0

DWord	Bit	Description
	29	**Monochrome Source Transparency Mode** This bit applies only when the source data is in monochrome. This bit determines whether or not the byte(s) at the destination corresponding to the pixel to which a given bit of the source data also corresponds will actually be written if that source data bit has the value of 0. This feature can make it possible to use the source as a transparency mask. The BLT Engine is configured to accepted either monochrome or color source data via the opcode field.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the source data. Wherever a bit in the source data has the value of 0, the color specified in the background color register is used as the source operand in the bit-wise operation for the pixel corresponding to the source data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1		Where a bit in the source data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the source data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

DWord	Bit	Description
	28	**Monochrome Pattern Transparency Mode** This bit applies only when the pattern data is monochrome. This bit determines whether or not

BR13 - BLT Raster OP, Control, and Destination Pitch

the byte(s) at the destination corresponding to the pixel to which a given bit of the pattern data also corresponds will actually be written if that pattern data bit has the value of 1. This feature can make it possible to use the pattern as a transparency mask. The BLT Engine is configured to accepted either monochrome or color pattern data via the opcode in the Opcode and Control register.

Value	Name	Description
0	**[Default]**	This causes normal operation with regard to the use of the pattern data. Where a bit in the pattern data has the value of 0, the color specified in the background color register is used as the pattern operand in the bit-wise operation for the pixel corresponding to the pattern data bit, and the bytes at the destination corresponding to that pixel are written with the result.
1		Wherever a bit in the pattern data has the value of 0, the byte(s) at the destination corresponding to the pixel to which the pattern data bit also corresponds are simply not written, and the data at those byte(s) at the destination are allowed to remain unchanged.

27:26 **32bpp Byte Mask**

This field is only used for 32bpp.

Value	Name
00b	**[Default]**
1xb	Write Alpha Channel
x1b	Write RGB Channel

25:24 **Color Depth**

Value	Name
00b	8 Bit Color Depth **[Default]**
01b	16 Bit Color Depth
10b	24 Bit Color Depth
11b	Reserved

23:16 **Raster Operation Select**

Default Value:	00000000b

These 8 bits are used to select which one of 256 possible raster operations is to be performed by the BLT Engine.

15:0 **Destination Pitch(Offset)**

These 16 bits store the signed memory address offset value by which the destination address originally specified in the Destination Address Register is incremented or decremented as each scan line's worth of destination data is written into the frame buffer by the BLT Engine, so that the destination address will point to the next memory address to which the next scan line's worth of destination data is to be written. If the intended destination of a BLT operation is within on-screen frame buffer memory, this offset is normally set so that each subsequent scan line's worth of destination data lines up vertically with the destination data in the scan line, above. However, if the intended destination of a BLT operation is within off-screen memory, this offset can be set

BR13 - BLT Raster OP, Control, and Destination Pitch		
		so that each subsequent scan line's worth of destination data is stored at a location immediately after the location where the destination data for the last scan line ended, in order to create a single contiguous block of bytes of destination data at the destination.

BR14 - Destination Width and Height

BR14 - Destination Width and Height		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
BR14 contains the values for the height and width of the data to be BLT. If these values are not correct, such that the BLT Engine is either expecting data it does not receive or receives data it did not expect, the system can hang.		
DWord	**Bit**	**Description**
0	31:29	**Reserved**
	28:16	**Destination Height** These 13 bits specify the height of the destination data in terms of the number of scan lines. This is a working register.
	15:13	**Reserved**
	12:0	**Destination Byte Width** These 13 bits specify the width of the destination data in terms of the number of bytes per scan line. The number of pixels per scan line into which this value translates depends upon the color depth to which the graphics system has been set.

BR15 - Color Pattern Address Lower order Address bits

BR15 - Color Pattern Address Lower order Address bits

Project:	CHV, BSW
Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:6	**Color Pattern Address** Format: GraphicsAddress[31:6] Lower 32bits of the 48bit addressing. There is no change to the Color Pattern address specification due to Non-Power-of-2 change. It remains the same as before. The pattern data must be located in linear memory. These 26 bits specify the starting address of the (8X8) pixel color pattern. The pattern data must be located on a pattern-size boundary. The pattern is always of 8x8 pixels, and therefore, its size is dependent upon its pixel depth. The pixel depth may be 8, 16, or 32 bits per pixel if the pattern is in color (the pixel depth of a color pattern must match the pixel depth to which the graphics system has been set). Monochrome patterns require 8 bytes and are applied through the instruction. Color patterns of 8, 16, and 32 bits per pixel color depth must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The Pattern Base Address programmed, must always be Cache Line (64byte) aligned.
	5:0	**Reserved** Format: MBZ

BR16 - Pattern Expansion Background and Solid Pattern Color

BR16 - Pattern Expansion Background and Solid Pattern Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR17 - Pattern Expansion Foreground Color

BR17 - Pattern Expansion Foreground Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome pattern data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR18 - Source Expansion Background and Destination Color

BR18 - Source Expansion Background and Destination Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Source Expansion Background Color Bits** These bits provide the one, two, or four bytes worth of color data that select the background color to be used in the color expansion of monochrome source data during BLT operations. This register is also used to support destination transparency mode and Solid color fill. Whether one, two, three, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR19 - Source Expansion Foreground Color

BR19 - Source Expansion Foreground Color		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:0	**Pattern/Source Expansion Foreground Color Bits** These bits provide the one, two, or four bytes worth of color data that select the foreground color to be used in the color expansion of monochrome source data during BLT operations. Whether one, two, or four bytes worth of color data is needed depends upon the color depth to which the BLT Engine has been set. For a color depth of 32bpp, 16bpp and 8bpp, bits [31:0], [15:0] and [7:0], respectively, are used.

BR27 - Destination Address Higher Order Address

BR27 - Destination Address Higher Order Address

Project:	CHV, BSW
Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Destination Address Upper DWORD** Format: GraphicsAddress[47:32] When tiling is enabled for XY-blits, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before. These upper 16bits of the 48bit address, along with BR09 register, will specify the starting pixel address of the destination data. This register is also the working destination address register for the upper 16bits of the destination address, and changes as the BLT Engine performs the accesses. Used as the scan line address (Destination Y Address and Destination Y1 Address) for BLT instructions: PIXEL_BLT, SCANLINE_BLT, and TEXT_BLT. In this case the address points to the first pixel in a scan line and is compared with the ClipRect Y1 and Y2 address registers to determine whether the scan line should be written or not. The Destination Y1 address is the top scan line to be written for text. Note that for non-XY blits (COLOR_BLT, SRC_COPY_BLT), this 16bits of the 48bit address, along with BR09 register, points to the first byte to be written. This register is always the last register written for a BLT drawing instruction. Writing BR27 starts the BLT engine execution. Note: Some instructions affect only one scan line (requiring only one coordinate); other instructions affect multiple scan lines and need both coordinates.

BR28 - Source Address Higher order Address

BR28 - Source Address Higher order Address		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:16	**Reserved** Format: MBZ
	15:0	**Source Address Upper DWORD** Format: GraphicsAddress[47:32] These upper 16bits of the 48bit address, specify the starting pixel address of the color or mono source data. When tiling is enabled for XY-blits with Color source surfaces, this base address should be limited to 4KB. Otherwise for XY blits, there is no restriction and it is same as before, including for monosource and text blits. Note that for non-XY blit with Color Source (SRC_COPY_BLT), this address points to the first byte to be read.

BR29 - Color Pattern Address Higher order Address

BR29 - Color Pattern Address Higher order Address		
Project:	CHV, BSW	
Source:	BlitterCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:16	**Reserved** Format: MBZ
	15:0	**Color Pattern Address Upper DWORD** Format: GraphicsAddress[47:32] These upper 16bits of the 48bit address,specify the starting address of the (8X8) pixel pattern.

BR30 - Setup Blit Color Pattern Address Higher Order Address

BR30 - Setup Blit Color Pattern Address Higher Order Address

Project:	CHV, BSW
Source:	BlitterCS
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Reserved** Format: MBZ
	15:0	**Setup Blit Color Pattern Address Upper DWORD** Format: GraphicsAddress[47:32] These upper 16bits of the 48bit address,specify the starting address of the (8X8) pixel pattern.

Byte Masked Media Block Message Header

MH_MBBM - Byte Masked Media Block Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Project: All Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** Must be DWord aligned (Bits 1:0 MBZ) for the write form of the message.
1	31:0	**Y Offset** Project: All Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Media Block Message Control** Project: All Format: MHC_MBBM_CONTROL [CHV, BSW] Specifies the Byte Masked message subtype and its additional input parameters.
3	31:0	**Byte Mask** Project: All Format: U32 Specifies the Byte Mask for writes when Message Mode field is BYTE_MASK. **Programming Notes** The Byte mask applies horizontally to each row of output: bit 0 for byte 0, through bit 31 for byte 31.
4	31:0	**FFTID** Project: All Format: MHC_FFTID [CHV, BSW] Fixed Function Thread ID
5-7	95:0	**Reserved**

MH_MBBM - Byte Masked Media Block Message Header

		Project:	All
		Format:	Ignore
		Ignored	

Byte Masked Media Block Message Header Control

MHC_MBBM_CONTROL - Byte Masked Media Block Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Project: All Format: Enumeration Specifies the Media Block Write Message subtype is Byte Masked. Value / Name / Description / Project: 02h / BYTE_MASK / The Block Height and Block Width fields are specified in this Dword. The Byte Mask qualifies which bytes are written. / All Others / Reserved / Reserved. / All
	29	**Reserved** Project: All Format: Ignore Ignored
	28:24	**Sub-Register Offset** Project: All Format: U5 This field is ignored (reserved) for Media Block Write message.
	23:22	**Reserved** Project: All Format: Ignore Ignored
	21:16	**Block Height** Project: All Format: U6 Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows Restriction: If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) *

MHC_MBBM_CONTROL - Byte Masked Media Block Message Header Control

		(# rows) <= 64 Dwords.
	15:10	**Reserved** Project: All Format: Ignore Ignored
	9:8	**Register Pitch Control** Project: All Format: U2 This field is ignored (reserved) for a Media Block Write message.
	7:6	**Reserved** Project: All Format: Ignore Ignored
	5:0	**Block Width** Project: All Format: U6 Width in bytes of the block being accessed. Range = [0,31] representing 1 to 32 Bytes. **Programming Notes** Must be DWord aligned for Media Block Write message.

CC_VIEWPORT

CC_VIEWPORT

Project:	CHV, BSW
Source:	PRM
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

The viewport state is stored as an array of up to 16 elements, each of which contains the DWords described here. The start of each element is spaced 2 DWords apart. The first element of the viewport state array is aligned to a 32-byte boundary. The Minimum Depth must be be greater than or equal to zero on D16_UNORM, D24_UNORM_X8_UINT, or D24_UNORM_S8_UINT depth formats. The Minimum Depth must be greater than or equal to -1.0 for D32_FLOAT_S8X24_UINT or D32_FLOAT formats. The Maximum Depth must be less than or equal to +1.0. The max must be greater than or equal to the min.

DWord	Bit	Description
0	31:0	**Minimum Depth** Project: All Format: IEEE_Float Indicates the minimum depth. The interpolated or computed depth is clamped to this value prior to the depth test. **Programming Notes** The Minimum depth value must be less-than-or-equal to the Maximum depth value. The Minimum depth value cannot be NAN (Not-A-Number). The Minimum depth value must not be less than -1.0.
1	31:0	**Maximum Depth** Project: All Format: IEEE_Float Indicates the maximum depth. The interpolated or computed depth is clamped to this value prior to the depth test. **Programming Notes** The Maximum depth value cannot be NAN (Not-A-Number). The Maximum depth value must be less-than-or-equal to +1.0.

Channel Mask Message Descriptor Control Field

MDC_CMASK - Channel Mask Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3:0	**Mask**

Project:	All
Format:	Enumeration

For the read message, indicates that which channels are read from the surface and included in the writeback message. For the write message, indicates which channels are included in the message payload and written to the surface.

Value	Name	Description	Project
00h	RGBA **[Default]**	Red, Green, Blue, and Alpha are included	All
01h	GBA	Green, Blue, and Alpha are included	All
02h	RBA	Red, Blue, and Alpha are included	All
03h	BA	Blue and Alpha are included	All
04h	RGA	Red, Green, and Alpha are included	All
05h	GA	Green and Alpha are included	All
06h	RA	Red and Alpha are included	All
07h	A	Alpha is included	All
08h	RGB	Red, Green, and Blue are included	All
09h	GB	Green and Blue are included	All
0Ah	RB	Red and Blue are included	All
0Bh	B	Blue is included	All
0Ch	RG	Red and Green are included	All
0Dh	G	Green is included	All
0Eh	R	Red is included	All
0Fh	Reserved	Ignored	All

Channel Mode Message Descriptor Control Field

MDC_CMODE - Channel Mode Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Channel Mode** Project: All Format: Enumeration Two modes of channel-enable are provided: a SIMD8 or SIMD16 Dword channel serial view of a register, and a SIMD4x2 view of a register.

Value	Name	Description	Project
0	Oword	All 4 Dwords are read or written if one or more of these channels are enabled	All
1	Dword	Each Dword is read or written only if its corresponding channel is enabled.	All

COLOR_CALC_STATE

COLOR_CALC_STATE

Project:	CHV, BSW
Source:	PRM
Size (in bits):	192
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This definition applies to [CHV, BSW] devices. It is pointed to by a field in 3DSTATE_CC_STATE_POINTERS, and stored at a 64-byte aligned boundary.

DWord	Bit	Description
0	31:24	**Stencil Reference Value** Project: CHV, BSW Format: U8.0 This field specifies the stencil reference value to compare against in the (front face) StencilTest function.
	23:16	**BackFace Stencil Reference Value** Project: CHV, BSW Format: U8.0 This field specifies the stencil reference value to compare against in the StencilTest function.
	15	**Round Disable Function Disable** Disables the round-disable function of the color calculator. Value 0 – Name: Cancelled – Description: Dithering is cancelled based on the data used by blend to avoid drift. Value 1 – Name: Not Cancelled – Description: Dithering is NOT cancelled.
	14:1	**Reserved** Format: MBZ
	0	**Alpha Test Format** This field selects the format for Alpha Reference Value and the format in which Alpha Test is performed. Value 0h – Name: ALPHATEST_UNORM8 – Description: UNorm8 Value 1h – Name: ALPHATEST_FLOAT32 – Description: Float32 **Programming Notes** Alpha-test format is independent of RT format. When PS outputs UNIT/SINT alpha-value, it will be treated as IEEE 32bit float number for the purpose of alpha-test.
1	31:0	**Alpha Reference Value As UNORM8**

COLOR_CALC_STATE

		Exists If: [Alpha Test Format] == 'ALPHATEST_UNORM8' Format: UNORM8 Upper 24 bits MBZ This field specifies the alpha reference value to compare against in the Alpha Test function.
	31:0	**Alpha Reference Value As FLOAT32** Exists If: [Alpha Test Format] == 'ALPHATEST_FLOAT32' Format: IEEE_Float This field specifies the alpha reference value to compare against in the Alpha Test function.
2	31:0	**Blend Constant Color Red** Format: IEEE_Float This field specifies the Red channel of the Constant Color used in Color Buffer Blending.
3	31:0	**Blend Constant Color Green** Format: IEEE_Float This field specifies the Green channel of the Constant Color used in Color Buffer Blending.
4	31:0	**Blend Constant Color Blue** Format: IEEE_Float This field specifies the Blue channel of the Constant Color used in Color Buffer Blending.
5	31:0	**Blend Constant Color Alpha** Format: IEEE_Float This field specifies the Alpha channel of the Constant Color used in Color Buffer Blending.

COLOR_PROCESSING_STATE - ACE State

COLOR_PROCESSING_STATE - ACE State

Project:	CHV, BSW
Source:	PRM
Size (in bits):	416
Default Value:	0x00000068, 0x4C382410, 0x9C887460, 0xEBD8C4B0, 0x604C3824, 0xB09C8874, 0x0000D8C4, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the ACE state used by the color processing function. It corresponds to DW29..DW41 of the Color Processing State.

DWord	Bit	Description
0	31:7	**Reserved** Format: MBZ
	6:2	**Skin Threshold** Format: U5 Used for Y analysis (min/max) for pixels which are higher than skin threshold. Value: 1-31 Value: 26 **[Default]**
	1	**Full Image Histogram** Default Value: 0 Format: Enable Used to ignore the area of interest for full image histogram.
	0	**ACE Enable** Format: Enable
1	31:24	**Y3** Default Value: 76 Format: U8 The value of the y_pixel for point 3 in PWL.
	23:16	**Y2** Default Value: 56 Format: U8 The value of the y_pixel for point 2 in PWL.
	15:8	**Y1**

COLOR_PROCESSING_STATE - ACE State

		Default Value: 36 Format: U8 The value of the y_pixel for point 1 in PWL.
	7:0	**Ymin** Default Value: 16 Format: U8 The value of the y_pixel for point 0 in PWL.
2	31:24	**Y7** Default Value: 156 Format: U8 The value of the y_pixel for point 7 in PWL.
	23:16	**Y6** Default Value: 136 Format: U8 The value of the y_pixel for point 6 in PWL.
	15:8	**Y5** Default Value: 116 Format: U8 The value of the y_pixel for point 5 in PWL.
	7:0	**Y4** Default Value: 96 Format: U8 The value of the y_pixel for point 4 in PWL.
3	31:24	**Ymax** Default Value: 235 Format: U8 The value of the y_pixel for point 11 in PWL.
	23:16	**Y10** Default Value: 216 Format: U8 The value of the y_pixel for point 10 in PWL.

COLOR_PROCESSING_STATE - ACE State

	15:8	**Y9** Default Value: 196 Format: U8 The value of the y_pixel for point 9 in PWL.
	7:0	**Y8** Default Value: 176 Format: U8 The value of the y_pixel for point 8 in PWL.
4	31:24	**B4** Default Value: 96 Format: U8 The value of the bias for point 4 in PWL.
	23:16	**B3** Default Value: 76 Format: U8 The value of the bias for point 3 in PWL.
	15:8	**B2** Default Value: 56 Format: U8 The value of the bias for point 2 in PWL.
	7:0	**B1** Default Value: 36 Format: U8 The value of the bias for point 1 in PWL.
5	31:24	**B8** Default Value: 176 Format: U8 The value of the bias for point 8 in PWL.
	23:16	**B7** Default Value: 156 Format: U8 The value of the bias for point 7 in PWL.

COLOR_PROCESSING_STATE - ACE State

	15:8	**B6** Default Value: 136 Format: U8 The value of the bias for point 6 in PWL.
	7:0	**B5** Default Value: 116 Format: U8 The value of the bias for point 5 in PWL.
6	31:16	**Reserved** Format: MBZ
	15:8	**B10** Default Value: 216 Format: U8 The value of the bias for point 10 in PWL.
	7:0	**B9** Default Value: 196 Format: U8 The value of the bias for point 9 in PWL.
7	31:27	**Reserved** Format: MBZ
	26:16	**S1** Format: U1.10 The value of the slope for point 1 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S0** Format: U1.10 The value of the slope for point 0 in PWL. The default is 1024/1024.
8	31:27	**Reserved** Format: MBZ
	26:16	**S3**

COLOR_PROCESSING_STATE - ACE State

		Format: U1.10 The value of the slope for point 3 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S2** Format: U1.10 The value of the slope for point 2 in PWL. The default is 1024/1024.
9	31:27	**Reserved** Format: MBZ
	26:16	**S5** Format: U1.10 The value of the slope for point 5 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S4** Format: U1.10 The value of the slope for point 4 in PWL. The default is 1024/1024.
10	31:27	**Reserved** Format: MBZ
	26:16	**S7** Format: U1.10 The value of the slope for point 7 in PWL. The default is 1024/1024.
	15:11	**Reserved** Format: MBZ
	10:0	**S6** Format: U1.10 The value of the slope for point 6 in PWL. The default is 1024/1024.
11	31:27	**Reserved** Format: MBZ
	26:16	**S9** Format: U1.10 The value of the slope for point 9 in PWL. The default is 1024/1024.

COLOR_PROCESSING_STATE - ACE State		
	15:11	**Reserved** Format: MBZ
	10:0	**S8** Format: U1.10 The value of the slope for point 8 in PWL. The default is 1024/1024.
12	31:11	**Reserved** Format: MBZ
	10:0	**S10** Format: U1.10 The value of the slope for point 10 in PWL. The default is 1024/1024.

COLOR_PROCESSING_STATE - CSC State

COLOR_PROCESSING_STATE - CSC State

Project: CHV, BSW

Source: PRM

Size (in bits): 288

Default Value: 0x00002000, 0x00000000, 0x00000400, 0x00000000, 0x000004B4, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the CSC state used by the color processing function. It corresponds to DW55..DW63 of the Color Processing State.

DWord	Bit	Description
0	31:29	**Reserved** Format: MBZ
	28:16	**C1** Default Value: 0 Format: S2.10 2's complement Transform coefficient
	15:3	**C0** Default Value: 1024 Format: S2.10 2's complement Transform coefficient
	2	**YUV_IN** Default Value: 0 Format: YUV CSC input offset enable.
	1	**YUV_OUT** Default Value: 0 Format: RGB CSC output offset enable.
	0	**Transform Enable** Format: Enable
1	31:26	**Reserved** Format: MBZ
	25:13	**C3**

COLOR_PROCESSING_STATE - CSC State

		Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C2** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
2	31:26	**Reserved** Format: MBZ
	25:13	**C5** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C4** Default Value: 1024 Format: S2.10 2's complement Transform coefficient.
3	31:26	**Reserved** Format: MBZ
	25:13	**C7** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
	12:0	**C6** Default Value: 0 Format: S2.10 2's complement Transform coefficient.
4	31:13	**Reserved** Format: MBZ
	12:0	**C8** Default Value: 1204 Format: S2.10 2's complement

COLOR_PROCESSING_STATE - CSC State

		Transform coefficient.
5	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 1** Default Value: 0 Format: S9 2's complement Offset Out for Y/R.
	9:0	**Offset In 1** Default Value: 0 Format: S9 2's complement Offset in for Y/R.
6	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 2** Default Value: 0 Format: S9 2's complement Offset out for U/G.
	9:0	**Offset in 2** Default Value: 0 Format: S9 2's complement Offset in for U/G.
7	31:20	**Reserved** Format: MBZ
	19:10	**Offset out 3** Default Value: 0 Format: S9 2's complement Offset out for V/B.
	9:0	**Offset in 3** Default Value: 0 Format: S9 2's complement Offset in for V/B.
8	31:17	**Reserved**

COLOR_PROCESSING_STATE - CSC State

		Format: MBZ
	16	**Alpha from State Select** Format: U1 Enumerated Type **Value / Name / Description** 0 — Alpha is taken from message 1 — Alpha is taken from state
	15:0	**Color Pipe Alpha** Format: U16

COLOR_PROCESSING_STATE - PROCAMP State

COLOR_PROCESSING_STATE - PROCAMP State

Project: CHV, BSW
Source: PRM
Size (in bits): 64
Default Value: 0x00020001, 0x01000000

This state structure contains the PROCAMP state used by the color processing function. It corresponds to DW53..DW54 of the Color Processing State.

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:17	**Contrast** Default Value: 1 Format: U4.7 Contrast magnitude.
	16:13	**Reserved** Format: MBZ
	12:1	**Brightness** Default Value: 0 Format: S7.4 2's complement Brightness magnitude.
	0	**PROCAMP Enable** Default Value: 1 Format: Enable
1	31:16	**Cos_c_s** Default Value: 256 Format: S7.8 2's complement UV multiplication cosine factor.
	15:0	**Sin_c_s** Default Value: 0 Format: S7.8 2's complement UV multiplication sine factor.

COLOR_PROCESSING_STATE - STD/STE State

COLOR_PROCESSING_STATE - STD/STE State

Project: CHV, BSW

Source: PRM

Size (in bits): 928

Default Value: 0x9A6E39F0, 0x400C0000, 0x00001180, 0xFE2F2E00, 0x000000FF, 0x00140000, 0xD82E0000, 0x8285ECEC, 0x00008282, 0x00000000, 0x02117000, 0xA38FEC96, 0x00008CC8, 0x00000000, 0x01478000, 0x0007C300, 0x00000000, 0x00000000, 0x1C180000, 0x00000000, 0x00000000, 0x00000000, 0x0007CF80, 0x00000000, 0x00000000, 0x1C080000, 0x00000000, 0x00000000, 0x00000000

This state structure contains the STD/STE state used by the color processing function.

DWord	Bit	Description
0	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate
	15:10	**Hue Max** Default Value: 14 Format: U6 Rectangle half width
	9:4	**Sat Max** Default Value: 31 Format: U6 Rectangle half length.
	3	**Reserved** Format: MBZ
	2	**Output Control** Value 0: Output Pixels **[Default]** Value 1: Output STD Decisions

COLOR_PROCESSING_STATE - STD/STE State

	1	**STE Enable** Format: Enable
	0	**STD Enable** Format: Enable
1	31	**Reserved** Format: MBZ
	30:28	**Diamond Margin** Default Value: 4 Format: U3
	27:21	**Diamond du** Default Value: 0 Format: S6 2's complement Rhombus center shift in the sat-direction, relative to the rectangle center.
	20:18	**HS Margin** Default Value: 3 Format: U3
	17:10	**Cos(α)** Format: S0.7 2's Compliment The default is 79/128
	9:8	**Reserved** Format: MBZ
	7:0	**Sin(α)** Format: S0.7 2's Compliment The default is 101/128
2	31:21	**Reserved** Format: MBZ
	20:13	**Diamond Alpha** Format: U2.6 1 / tan(β) The default is 100/64
	12:7	**Diamond Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.

COLOR_PROCESSING_STATE - STD/STE State

	6:0	**Diamond dv** Default Value: 0 Format: S6 2's complement
3	31:24	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	23:16	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	15:8	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
	7	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	6:0	**Reserved** Format: MBZ
4	31:18	**Reserved** Format: MBZ
	17:13	**Y_Slope_2** Format: U2.3 Slope between points Y3 and Y4. The default is 31/8.
	12:8	**Y_Slope_1** Format: U2.3 Slope between points Y1 and Y2. The default is 31/8.
	7:0	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function

COLOR_PROCESSING_STATE - STD/STE State

5	31:16	**INV_skin_types_margin** Format: U0.16 1/(2* Skin_types_margin) Value: 20 — Name: **[Default]** — Description: Skin_Type_margin
	15:0	**Inverse Margin VYL** Format: U0.16 1 / Margin_VYL The default is 3300/65536
6	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**Inverse Margin VYU** Format: U0.16 1 / Margin_VYU The default is 1600/65536.
7	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L**

COLOR_PROCESSING_STATE - STD/STE State

		Default Value: 236 Format: U8 Y point 2 of the lower part of the detection PWLF.
8	31:27	**Reserved** Format: MBZ
	26:16	**S0L** Format: S2.8 2's complement Slope 0 of the lower part of the detection PWLF. The default is -5/256.
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8 V Bias 2 of the lower part of the detection PWLF.
9	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Format: S2.8 2's complement Slope 2 of the lower part of the detection PWLF. The default is 0/256.
	10:0	**S1L** Format: S2.8 2's complement Slope 1 of the lower part of the detection PWLF. The default is 0/256.
10	31:27	**Reserved** Format: MBZ
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U**

COLOR_PROCESSING_STATE - STD/STE State

		Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Format: S2.8 2's complement Slope 3 of the lower part of the detection PWLF. The default is 0/256.
11	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
12	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Format: S2.8 2's complement Slope 0 of the upper part of the detection PWLF. The default is 256/256.
	15:8	**B3U** Default Value: 140 Format: U8 V Bias 3 of the upper part of the detection PWLF.

COLOR_PROCESSING_STATE - STD/STE State

	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
13	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Format: S2.8 2's complement Slope 2 of the upper part of the detection PWLF. The default is -179/256.
	10:0	**S1U** Format: S2.8 2's complement Slope 1 of the upper part of the detection PWLF. The default is -113/256.
14	31:28	**Reserved** Format: MBZ
	27:20	**Skin Types Margin** Default Value: 20 Format: U8 Skin types Y margin.
	19:12	**Skin Types Thresh** Default Value: 120 Format: U8 Skin types Y threshold.
	11	**Skin Type Enable** Format: Enable Treat differently bright and dark skin types. Value / Name / Description 0 / **[Default]** / Disable
	10:0	**S3U** Format: S2.8 2's complement Slope 3 of the upper part of the detection PWLF. The default is 0/256.
15	31	**Reserved** Format: MBZ

COLOR_PROCESSING_STATE - STD/STE State

	30:21	**SATB1** Format: S7.2 2's complement First bias for the saturation PWLF (bright skin). The default is -8/4.
	20:14	**SATP3** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (bright skin).
	13:7	**SATP2** Default Value: 6 Format: S6 2's complement Second point for the saturation PWLF (bright skin).
	6:0	**SATP1** Format: S6 2's complement First point for the saturation PWLF (bright skin). The default is -6.
16	31	**Reserved** Format: MBZ
	30:20	**SATS0** Format: U3.8 Zeroth slope for the saturation PWLF (bright skin). The default is 297/256.
	19:10	**SATB3** Format: S7.2 2's complement Third bias for the saturation PWLF (bright skin). The default is 124/4.
	9:0	**SATB2** Format: S7.2 2's complement Second bias for the saturation PWLF (bright skin). The default is 8/4.
17	31:22	**Reserved** Format: MBZ
	21:11	**SATS2** Format: U3.8 Second slope for the saturation PWLF (bright skin). The default is 297/256.
	10:0	**SATS1**

COLOR_PROCESSING_STATE - STD/STE State

		Format: U3.8 First slope for the saturation PWLF (bright skin). The default is 85/256.
18	31:25	**HUEP3** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (bright skin)
	24:18	**HUEP2** Default Value: 6 Format: S6 2's complement Second point for the hue PWLF (bright skin)
	17:11	**HUEP1** Format: S6 2's complement First point for the hue PWLF (bright skin). The default is -6.
	10:0	**SATS3** Format: U3.8 Thrid slope for the saturation PWLF (bright skin). The default is 256/256.
19	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3** Format: S7.2 2's complement Third bias for the hue PWLF (bright skin). The default is 56/4.
	19:10	**HUEB2** Format: S7.2 2's complement Second bias for the hue PWLF (bright skin). The default is 8/4.
	9:0	**HUEB1** Format: S7.2 2's complement First bias for the hue PWLF (bright skin). The default is -8/4.
20	31:22	**Reserved** Format: MBZ
	21:11	**HUES1** Format: U3.8

COLOR_PROCESSING_STATE - STD/STE State

		First slope for the hue PWLF (bright skin) The default is 85/256.
	10:0	**HUES0** Format: U3.8 Zeroth slope for the hue PWLF (bright skin) The default is 384/256.
21	31:22	**Reserved** Format: MBZ
	21:11	**HUES3** Format: U3.8 Third slope for the hue PWLF (bright skin) The default is 256/256.
	10:0	**HUES2** Format: U3.8 Second slope for the hue PWLF (bright skin) The default is 384/256.
22	31	**Reserved** Format: MBZ
	30:21	**SATB1_DARK** Format: S7.2 2's complement First bias for the saturation PWLF (dark skin) The default is 0/4.
	20:14	**SATP3_DARK** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (dark skin)
	13:7	**SATP2_DARK** Default Value: 31 Format: S6 2's complement Second point for the saturation PWLF (dark skin)
	6:0	**SATP1_DARK** Format: S6 2's complement First point for the saturation PWLF (dark skin). The default is -11.
23	31	**Reserved** Format: MBZ
	30:20	**SATS0_DARK**

COLOR_PROCESSING_STATE - STD/STE State

		Format: U3.8 Zeroth slope for the saturation PWLF (dark skin). The default is 397/256.
	19:10	**SATB3_DARK** Format: S7.2 2's complement Third bias for the saturation PWLF (dark skin). The default is 124/4.
	9:0	**SATB2_DARK** Format: S7.2 2's complement Second bias for the saturation PWLF (dark skin). The default is 124/4.
24	31:22	**Reserved** Format: MBZ
	21:11	**SATS2_DARK** Format: U3.8 Second slope for the saturation PWLF (dark skin). The default is 256/256.
	10:0	**SATS1_DARK** Format: U3.8 First slope for the saturation PWLF (dark skin). The default is 189/256.
25	31:25	**HUEP3_DARK** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	24:18	**HUEP2_DARK** Default Value: 2 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	17:11	**HUEP1_DARK** Default Value: 0 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	10:0	**SATS3_DARK** Format: U3.8 Third slope for the saturation PWLF (dark skin). The default is 256/256.

COLOR_PROCESSING_STATE - STD/STE State

26	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3_DARK** Format: S7.2 2's complement Third bias for the hue PWLF (dark skin). The default is 56/4.
	19:10	**HUEB2_DARK** Format: S7.2 2's complement Second bias for the hue PWLF (dark skin). The default is 0/4.
	9:0	**HUEB1_DARK** Format: S7.2 2's complement First bias for the hue PWLF (dark skin). The default is 0/4.
27	31:22	**Reserved** Format: MBZ
	21:11	**HUES1_DARK** Format: U3.8 First slope for the hue PWLF (dark skin). The default is 0/256.
	10:0	**HUES0_DARK** Format: U3.8 Zeroth slope for the hue PWLF (dark skin). The default is 256/256.
28	31:22	**Reserved** Format: MBZ
	21:11	**HUES3_DARK** Format: U3.8 Third slope for the hue PWLF (dark skin). The default is 256/256.
	10:0	**HUES2_DARK** Format: U3.8 Second slope for the hue PWLF (dark skin). The default is 299/256.

COLOR_PROCESSING_STATE - TCC State

COLOR_PROCESSING_STATE - TCC State

Project:	CHV, BSW
Source:	PRM
Size (in bits):	352
Default Value:	0xDCDCDC00, 0xDCDCDC00, 0x1E34CC91, 0x3E3CCE91, 0x02E80195, 0x0197046B, 0x01790174, 0x00096000, 0x00000000, 0x03030000, 0x009201C0

This state structure contains the TCC state used by the color processing function. It corresponds to DW42..DW52 of the Color Processing State.

DWord	Bit	Description
0	31:24	**SatFactor3** Default Value: 220 Format: U1.7 The saturation factor for yellow.
	23:16	**SatFactor2** Default Value: 220 Format: U1.7 The saturation factor for red.
	15:8	**SatFactor1** Default Value: 220 Format: U1.7 The saturation factor for magenta.
	7	**TCC Enable** Format: Enable
	6:0	**Reserved** Format: MBZ
1	31:24	**SatFactor6** Default Value: 220 Format: U1.7 The saturation factor for blue.
	23:16	**SatFactor5** Default Value: 220 Format: U1.7 The saturation factor for cyan.

COLOR_PROCESSING_STATE - TCC State

	15:8	**SatFactor4** Default Value: 220 Format: U1.7 The saturation factor for green.
	7:0	**Reserved** Format: MBZ
2	31:30	**Reserved** Format: MBZ
	29:20	**Base Color 3** Default Value: 483 Format: U10
	19:10	**Base Color 2** Default Value: 307 Format: U10
	9:0	**Base Color 1** Default Value: 145 Format: U10
3	31:30	**Reserved** Format: MBZ
	29:20	**Base Color 6** Default Value: 995 Format: U10
	19:10	**Base Color 5** Default Value: 819 Format: U10
	9:0	**Base Color 4** Default Value: 657 Format: U10
4	31:16	**Color Transit Slope 23** Default Value: 744 Format: U0.16 The calculation result of 1 / (BC3 - BC2) [1/62]
	15:0	**Color Transit Slope 12**

COLOR_PROCESSING_STATE - TCC State

		Default Value: 405 Format: U0.16 The calculation result of 1 / (BC2 - BC1) [1/57]
5	31:16	**Color Transit Slope 45** Default Value: 407 Format: U0.16 The calculation result of 1 / (BC5 - BC4) [1/57]
	15:0	**Color Transit Slope 34** Default Value: 1131 Format: U0.16 The calculation result of 1 / (BC4 - BC3) [1/61]
6	31:16	**Color Transit Slope 61** Default Value: 377 Format: U0.16 The calculation result of 1 / (BC1 - BC6) [1/62]
	15:0	**Color Transit Slope 56** Default Value: 372 Format: U0.16 The calculation result of 1 / (BC6 - BC5) [1/62]
7	31:22	**Color Bias 3** Default Value: 0 Format: U2.8 Color bias for BaseColor3.
	21:12	**Color Bias 2** Default Value: 150 Format: U2.8 Color bias for BaseColor2.
	11:2	**Color Bias 1** Default Value: 0 Format: U2.8 Color bias for BaseColor1.

COLOR_PROCESSING_STATE - TCC State

	1:0	**Reserved** Format: MBZ
8	31:22	**Color Bias 6** Default Value: 0 Format: U2.8 Color bias for BaseColor6.
	21:12	**Color Bias 5** Default Value: 0 Format: U2.8 Color bias for BaseColor5.
	11:2	**ColorBias4** Default Value: 0 Format: U2.8 Color bias for BaseColor4.
	1:0	**Reserved** Format: MBZ
9	31	**Reserved** Format: MBZ
	30:24	**UV Threshold** Default Value: 3 Format: U7 Low UV threshold.
	23:19	**Reserved** Format: MBZ
	18:16	**UV Threshold Bits** Default Value: 3 Format: U3 Low UV transition width bits.
	15:13	**Reserved** Format: MBZ
	12:8	**STE Threshold** Default Value: 0 Format: U5

COLOR_PROCESSING_STATE - TCC State

		Skin tone pixels enhancement threshold.
	7:3	**Reserved** Format: MBZ
	2:0	**STE Slope Bits** Default Value: 0 Format: U3 Skin tone pixels enhancement slope bits.
10	31:16	**Inverse UVMax Color** Default Value: 146 Format: U0.16 1 / UVMaxColor. Used for the SFs2 calculation.
	15:9	**Reserved** Format: MBZ
	8:0	**UVMax Color** Default Value: 448 Format: U9 The maximum absolute value of the legal UV pixels. Used for the SFs2 calculation.

Color Calculator State Pointer Message Header Control

MHC_RT_CCSP - Color Calculator State Pointer Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:6	**Color Calculator State Pointer** Project: All Format: GeneralStateOffset[31:6] Specifies the 64-byte aligned point to the color calculator state. This pointer is relative to the General State Base Address.
	5:0	**Reserved** Project: All Format: Ignore Ignored

Color Code Message Header Control

MHC_RT_CC - Color Code Message Header Control		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	32	
Default Value:	0x00000000	

DWord	Bit	Description
0	31:10	**Reserved** Project: All Format: Ignore Ignored
	9:8	**Color Code** Project: All Format: U2 This ID is assigned by the Windower unit and is used to track synchronizing events. Reserved for HW implementation use
	7:0	**FFTID** Project: All Format: U8 This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.

Context Descriptor Format

Context Descriptor Format

Project:	CHV, BSW
Source:	PRM
Size (in bits):	64
Default Value:	0x00000020, 0x00000000

This is the format of context descriptors which make up submitted execlists.

DWord	Bit	Description

0	63:32	**Context ID**
		Context ID is a unique field assigned by GFX driver when a new context is created by which it is identified across all hierarchies of SW and HW.
		• Context ID is used for semaphore signaling by hardware and software.
		• Context ID matching is used by hardware to detect Lite Restore.
		• Context ID is used by hardware for page fault reporting and response with IOMMU.
		• Context switch reason and the associated Context ID are reported to Context Switch Status Buffer by hardware on a context switch.
		
		Context ID which is a 32 bit field is further divided in to three segments described below:
		• **Bits[63:55] (Bits 31:23 of Context ID)** is referred to as GroupID. GroupId+PASID combination of a context must be a unique identifier for contexts that are active in the system. The definition of active context is listed as:
		  • Any Context that is already submitted to h/w or already running in h/w.
		  • Any Context that hit page faults, was preempted (didn't run to context complete), and is waiting to be resubmitted pending IOMMU "last in group" response.
		  • Any Context that has experienced reset but not all faults are responded to.
		• **Bit[54] (Bit 22 of Context ID)** – MBZ for SW programming; this bit is used by hardware to distinguish between F&H vs F&S page requests and response messages to and from IOMMU. This bit is used by hardware on receiving page response to properly manage the page fault counters
		• **Bit[53] (Bit 21 of Context ID)** – MBZ from SW programming, is reserved for future hardware use.
		• **Bits[52:32] (Bits 20:0 of Context ID)** are for software use-only and must be unique field assigned by GFX driver when a new context is created.

	31:12	**Logical Ring Context Address (LRCA)** Format: GraphicsAddress[31:12] This field contains the 4 KB-aligned address of the Logical Ring Context associated with this execlist element. LRCA must be always programmed in GGTT memory.
	11:9	**Reserved**

Context Descriptor Format

	Format:	MBZ

8 Privilege Access

This field when set indicates PPGTT enabled in legacy context mode.
In advanced context mode this field is reserved and must be zero.

7:6 Fault Handling

Project:	CHV, BSW
Source:	RenderCS

Value	Name	Description	Project
0h	Fault and Hang	Fault model is not supported and fault occurrence is treated as catastrophic. GAM indicates Fault Error to Command streamer. Fault Error interrupt is reported to scheduler. Command Streamer will not initiate context switch on occurrence of Fault Error.	
1h	Reserved	Reserved	
2h	Reserved	Reserved	
3h	Reserved		

Programming Notes	Project
When execlist mode is set to "Legacy Context mode" Fault Handling mode must be set to "Fault and Hang." For proper programming for Page Fault modes, refer to memory interface section of the PRM for the corresponding generation.	CHV, BSW

7:6 Reserved

Project:	CHV, BSW
Source:	BlitterCS, VideoCS, VideoCS2, VideoEnhancementCS
Format:	MBZ

5 L3-LLC Coherency

Project:	CHV, BSW

The L3-LLC Coherency is controlled by HDC_Chicken0::ForceNonCoherent (bit 4). The L3-LLC Coherency bit must be set for the ForceNonCoherent control to work properly.

Value	Name
1h	Required **[Default]**
0h	Reserved

4:3 Addressing Mode & Legacy Context

Project:	CHV, BSW
Format:	U2

Legacy context set indicates GPU is operating in legacy context mode of operation and doesn't support any SVM features. Legacy context reset indicates GPU is operating in advanced context mode of operation and support SVM features. Based on the Context mode set Addressing mode is interpreted appropriately. The table below summarizes the combinations supported.

Context Descriptor Format

GFX engine always uses 32b virtual addressing mode when translated using GGTT irrespective of below options.

Value	Name	Description
00b	Advanced Context with no A/D support	GPU is enabled for advanced context mode and supports SVM features. GPU DOESN'T support Access and Dirty bit management in page tables. GPU supports 64b(48bit canonical) PPGTT graphics virtual addressing. PDP0_DESCRIPTOR contains the PASID (process address space identifier) and other PDP Descriptors are ignored.
01b	Legacy Context with no 64 bit VA support	GPU is enabled for legacy context mode of operation and DOESN'T support any SVM features. GPU supports 32b PPGTT graphics virtual addressing. PDP*_DESCRIPTOR contains the base address to 4GB of memory space supported.
10b	Advanced Context with A/D support	GPU is enabled for advanced context mode and supports SVM features. GPU DOES support Access and Dirty bit management in page tables. GPU supports 64b (48bit canonical) PPGTT graphics virtual addressing. PDP0_DESCRIPTOR contains the PASID (process address space identifier) and other PDP Descriptors are ignored.
11b	Legacy Context with 64 bit VA support	GPU is enabled for legacy context mode of operation and DOESN'T support any SVM features. GPU supports 64b (48bit canonical) PPGTT graphics virtual addressing and PDP0_DESCRIPTOR contains the base address to PML4 and other PDP Descriptors are ignored.

Programming Notes	Project
Advanced context and 64 bit Virtual addressing are not supported on CHV, BSW. Only legal value that must be programmed on CHV, BSW is "01b: Legacy Context with no 64 bit VA support".	CHV, BSW

2 **Force Restore**

Setting this bit will force a context restore operation when switching to this context even if the LRCA in the CCID register (normally the LRCA of the last context from the prior execlist) matches this one.

Note that it is legal (and likely desirable) for the **Render Context Restore Inhibit** bit (part of the CTXT_SR_CTL register) in the context image being restored to also be set. The "ring" context is being forced to be restored from a newly initialized context despite a possible LRCA match. However, the render context for such a newly initialized context will likely be uninitialized and so should not be restored.

Programming Notes	Project	Source
Force Restore bit must be always be set on all context submissions, when SW intends to use semaphore signaling (MI_SEMAPHORE_SIGNAL) between command streamers, this is to address known HW issue.	CHV, BSW	PRMPRM

1 **Force PD Restore**

Setting this bit will cause the on-chip page directory to be reloaded from the PD image in memory even on an LRCA match. No other operations of context restore will occur on an LRCA match, however. Software should set this bit if it has updated a context's page directory and

Context Descriptor Format		
		wants the context to begin using the new page directory without having to switch away from it (to another context) and back again. Setting this bit will have no effect if **Force Restore** is also set; a complete context restore (including the PD) will be performed.
	0	**Valid** Set if this register holds a valid context descriptor. SW should set this bit in the Element registers that it has set up to contain valid context descriptors. Any execlist elements that are not used in a submitted execlist must have this bit clear.

Context Status

Context Status

Project: CHV, BSW
Source: PRM
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0	63:32	**Context ID** Format: U32
	31:25	**Reserved** Format: MBZ
	24:20	**Reserved** Project: Format: MBZ
	19:16	**Reserved** Project: CHV, BSW Format: MBZ
	19:16	**Display Plane** Project: CHV, BSW This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set. Value / Name: 0h: Reserved (future Sprite A) 1h: Reserved (future Sprite B) 2h: Reserved (future Sprite C) 3h: Display Plane Sprite A2 4h: Display Plane Sprite B2 5h: Display Plane Sprite C2 6h: Display Plane Sprite A3 7h: Display Plane Sprite B3 8h: Display Plane Sprite C3 [9h, Fh]: Reserved
	15	**Lite Restore** Format: Enable This bit is only valid only when Preempted bit is set. When set, this bit indicates that a given

Context Status

		context got preempted with the same context resulting in Lite Restore in HW.
	14:12	**Display Plane** Project: CHV, BSW This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the "Wait on Scanline" or "Wait on Vblnak" or "Wait on sync Flip" is set. (Future - could remove the Sprites and move to bits 19:16) 0h: Display Plane-A 1h: Display Plane-B 2h: Display Plane-C 3h: Display Plane Sprite A 4h: Display Plane Sprite B 5h: Display Plane Sprite C
	11	**Semaphore Wait Mode** 0h: Signal Mode 1h: Poll Mode
	10:9	**Reserved** Format: MBZ
	8	**Wait on Scanline**
	7	**Wait on Semaphore**
	6	**Wait on V-blank**
	5	**Wait on Sync Flip**
	4	**Context Complete** Element is completely processed (Head eqv to Tail) and resulted in a context switch.
	3	**ACTIVE to IDLE** Following this context switch there is no active element available in HW to execute
	2	**Element Switch** Context Switch happened from first element in the current execlist to the second element of the same execlist
	1	**Preempted** Submission of a new execlist has resulted in context switch. The switch is from element in current execlist to element in pending execlist
	0	**IDLE to ACTIVE** Execlist submitted when HW is IDLE. When this bit is set rest of the fields in CSQ are not valid.

Data Port 0 Message Types

MT_DP0 - Data Port 0 Message Types

Project:	CHV, BSW
Source:	DataPort 0
Size (in bits):	5
Default Value:	0x00000000

Lists all the Message Types in a Data Port 0 Message Descriptor [18:14]. The Legacy messages are encoded in Data Port 0 with Bit 18 set to zero. The Message Header is optional for many (but not all) of these operations. The Scratch Block messages are encoded in Data Port 0 with Bit 18 set to one. A Message Header is required.

DWord	Bit	Description
0	4	**Legacy DAP-DC Message** Format: Enumeration Legacy Message

Value	Name	Description
0h	No **[Default]**	Legacy DAP-DC Message
1h	Reserved	Scratch Block Message, descriptor uses different Message Type encoding

DWord	Bit	Description
	3:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description	Project
00h	MT0R_OWB **[Default]**	Oword Block Read message	
01h	MT0R_OWUB	Unaligned Oword Block Read message	
02h	MT0R_OWDB	Oword Dual Block Read message	
03h	MT0R_DWS	Dword Scattered Read message	
04h	MT0R_BS	Byte Scattered Read message	
07h	MT0_MEMFENCE	Memory Fence message	
08h	MT0W_OWB	Oword Block Write message	
0Ah	MT0W_OWDB	Oword Dual Block Write message	
0Bh	MT0W_DWS	Dword Scattered Write message	
0Ch	MT0W_BS	Byte Scattered Write message	
Others	Reserved	Ignored	

Data Port 1 Message Types

MT_DP1 - Data Port 1 Message Types

Project: CHV, BSW
Source: DataPort 1
Size (in bits): 5
Default Value: 0x00000000

Lists all the Message Types in a Data Port 1 Message Descriptor [18:14]. Most surface and atomic operations, both typed and untyped, are encoded on Data Port 1. The Message Header is optional for many (but not all) of these operations. Most A64 Stateless operations are also encoded on Data Port 1. The Message Header is forbidden for all A64 messages on Data Port 1.

DWord	Bit	Description
0	4:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description	Project
00h	MT1R_T **[Default]**	Transpose Read message	
01h	MT1R_US	Untyped Surface Read message	
02h	MT1A_UI	Untyped Atomic Integer Operation message	
03h	MT1A_UI4x2	Untyped Atomic Integer Operation SIMD4x2 message	
04h	MT1R_MB	Media Block Read message	
05h	MT1R_TS	Typed Surface Read message	
06h	MT1A_TA	Typed Atomic Integer Operation message	
07h	MT1A_TA4x2	Typed Atomic Integer Operation SIMD4x2 message	
08h	Reserved	Ignored	CHV, BSW
09h	MT1W_US	Untyped Surface Write mesage	
0Ah	MT1W_MB	Media Block Write message	
0Bh	MT1A_TC	Typed Atomic Counter Operation message	
0Ch	MT1A_TC4x2	Typed Atomic Counter Operation SIMD4x2 message	
0Dh	MT1W_TS	Typed Surface Write message	
0Eh	Reserved	Ignored	CHV, BSW
10h	MT1R_A64_SB	A64 Scattered Read message	
11h	MT1R_A64_US	A64 Untyped Surface Read message	
12h	MT1A_A64_UI	A64 Untyped Atomic Integer Operation message	
13h	MT1A_A64_UI4x2	A64 Untyped Atomic Integer Operation SIMD4x2	

MT_DP1 - Data Port 1 Message Types

			message		
		14h	MT1R_A64_B	A64 Block Read message	
		15h	MT1W_A64_B	A64 Block Write message	
		18h	Reserved	Ignored	CHV, BSW
		19h	MT1W_A64_US	A64 Untyped Surface Write message	
		1Ah	MT1W_A64_SB	A64 Scattered Write message	
		1Bh	MT1A_UF	Untyped Atomic Float Operation message	
		1Ch	MT1A_UF4x2	Untyped Atomic Float Operation SIMD4x2 message	
		1Dh	MT1A_A64_UF	A64 Untyped Atomic Float Operation message	
		1Eh	MT1A_A64_UF4x2	A64 Untyped Atomic Float Operation SIMD4x2 message	
		Others	Reserved	Ignored	

Data Size Message Descriptor Control Field

MDC_DS - Data Size Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**Data Size** Project: All Format: Enumeration Specifies the number of Bytes to be read or written

Value	Name	Description	Project
00h	B	1 Byte	All
01h	W	2 Bytes	All
02h	DW	4 Bytes	All
03h	Reserved	Reserved	All

DstRegNum

DstRegNum

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	8
Default Value:	0x00000000

Description	Project
Register Number The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number. An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers. RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands.	CHV, BSW

DWord	Bit	Description
0	7:0	**Destination Register Number** Value / Name / Description: 0-127 / If {Dst/Src0/Src1/Src2}.RegFile==GRF / 0-0ffh / If {Dst/Src0/Src1/Src2}.RegFile==ARF / This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

DstSubRegNum

DstSubRegNum	
Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	5
Default Value:	0x00000000

Description	Project
Subregister Number The subregister number for the operand. For a GRF register, is the byte address within a 256-bit (32-byte) register. For an ARF register, determines the sub-register number according to the specified encoding for the given architecture register. RegNum and SubRegNum together provide the byte-aligned address for the origin of a GRF register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero.	CHV, BSW

Programming Notes
Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

DWord	Bit	Description
0	4:0	**Destination Sub Register Number** Value / Name / Description: 0-31 / If {Dst/Src0/Src1/Src2}.RegFile==GRF / 0-0ffh / If {Dst/Src0/Src1/Src2}.RegFile==ARF / This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

Dword Data Payload Register

MDCR_DW - Dword Data Payload Register

Project: CHV, BSW
Source: PRM
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Dword0** Project: All Format: U32 Specifies the slot 0 data in this payload register
0.1	31:0	**Dword1** Project: All Format: U32 Specifies the slot 1 data in this payload register
0.2	31:0	**Dword2** Project: All Format: U32 Specifies the slot 2 data in this payload register
0.3	31:0	**Dword3** Project: All Format: U32 Specifies the slot 3 data in this payload register
0.4	31:0	**Dword4** Project: All Format: U32 Specifies the slot 4 data in this payload register
0.5	31:0	**Dword5** Project: All Format: U32 Specifies the slot 5 data in this payload register

MDCR_DW - Dword Data Payload Register		
0.6	31:0	**Dword6** Project: All Format: U32 Specifies the slot 6 data in this payload register
0.7	31:0	**Dword7** Project: All Format: U32 Specifies the slot 7 data in this payload register

Dword SIMD4x2 Atomic CMPWR Message Data Payload

MDP_AOP4X2_DW2 - Dword SIMD4x2 Atomic CMPWR Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Src0 Slot0** Format: U32|S31|F32 Specifies the Slot 0 Source 0 data
1	31:0	**Src1 Slot0** Format: U32|S31|F32 Specifies the Slot 0 Source 1 data
2-3	63:0	**Reserved** Format: Ignore Ignored
4	31:0	**Src0 Slot1** Format: U32|S31|F32 Specifies the Slot 1 Source 0 data
5	31:0	**Src1 Slot1** Format: U32|S31|F32 Specifies the Slot 1 Source 1 data
6-7	63:0	**Reserved** Format: Ignore Ignored

Dword SIMD4x2 Atomic Operation Message Data Payload

MDP_AOP4X2_DW1 - Dword SIMD4x2 Atomic Operation Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Dword0** Format: U32|S31|F32 Specifies the Slot 0 Source or Return data
1-3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**Dword1** Format: U32|S31|F32 Specifies the Slot 1 Source or Return data
5-7	95:0	**Reserved** Format: Ignore Ignored

Dword SIMD4x2 Data Payload

MDP_DW_SIMD4X2 - Dword SIMD4x2 Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Red Slot0** Project: All Format: U32 Specifies the Slot 0 red channel data
1	31:0	**Green Slot0** Project: All Format: U32 Specifies the Slot 0 green channel data
2	31:0	**Blue Slot0** Project: All Format: U32 Specifies the Slot 0 blue channel data
3	31:0	**Alpha Slot0** Project: All Format: U32 Specifies the Slot 0 alpha channel data
4	31:0	**Red Slot1** Project: All Format: U32 Specifies the Slot 1 red channel data
5	31:0	**Green Slot1** Project: All Format: U32 Specifies the Slot 1 green channel data

MDP_DW_SIMD4X2 - Dword SIMD4x2 Data Payload		
6	31:0	**Blue Slot1** Project: All Format: U32 Specifies the Slot 1 blue channel data
7	31:0	**Alpha Slot1** Project: All Format: U32 Specifies the Slot 1 alpha channel data

Dword SIMD8 Atomic Operation CMPWR Message Data Payload

MDP_AOP8_DW2 - Dword SIMD8 Atomic Operation CMPWR Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Slot [7:0] Source 0 data
1.0-1.7	255:0	**Src1** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Slot [7:0] Source 1 data

Dword SIMD8 Data Payload

MDP_DW_SIMD8 - Dword SIMD8 Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Data[7:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Slot [7:0] data

Dword SIMD16 Atomic Operation CMPWR Message Data Payload

MDP_AOP16_DW2 - Dword SIMD16 Atomic Operation CMPWR Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0[7:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Source 0 data for Slot [7:0]
1.0-1.7	255:0	**Src0[15:8]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Source 0 data for Slot [15:8]
2.0-2.7	255:0	**Src1[7:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Source 1 data for Slot [7:0]
3.0-3.7	255:0	**Src1[15:8]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Source 1 data for Slot [15:8]

Dword SIMD16 Data Payload

MDP_DW_SIMD16 - Dword SIMD16 Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[7:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Slot [7:0] data
1.0-1.7	255:0	**Data[15:8]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the Slot [15:8] data

DX9_CONSTANTB_ENTRY

DX9_CONSTANTB_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTB_* commands. Each entry provides the values for the one boolean constant being updated.		
DWord	**Bit**	**Description**
0	31:0	**Component** Format: U32 The boolean value to be stored.

DX9_CONSTANTF_ENTRY

DX9_CONSTANTF_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTF_* commands. Each entry provides the values for the four components of one float constant being updated.		
DWord	**Bit**	**Description**
0	127:96	**Component 3** Format: IEEE_Float The 4th component of the nth float to be stored.
	95:64	**Component 2** Format: IEEE_Float The 3rd component of the nth float to be stored.
	63:32	**Component 1** Format: IEEE_Float The 2nd component of the nth float to be stored.
	31:0	**Component 0** Format: IEEE_Float The 1st component of the nth float to be stored.

DX9_CONSTANTI_ENTRY

DX9_CONSTANTI_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
This structure is the payload of the 3DSTATE_DX9_CONSTANTI_* commands. Each entry provides the values for the four components of one integer constant being updated.		
DWord	**Bit**	**Description**
0	31:0	**Component 0** Format: U32 The 1st component of the nth float to be stored.
1	31:0	**Component 1** Format: U32 The 2nd component of the nth float to be stored.
2	31:0	**Component 2** Format: U32 The 3rd component of the nth float to be stored.
3	31:0	**Component 3** Format: U32 The 4th component of the nth float to be stored.

Encoder Statistics Format

Encoder Statistics Format

Project:	CHV, BSW
Source:	VideoEnhancementCS
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

The per block data is intended for use by the video encoder and consists of 16 bytes of Denoise block data and FMD variances. Much of the data is encoded as an 8-bit mantissa with the leading 1 removed and a 4-bit shift. To recover the original 17-bit integer this code can be used: If (exp != 0) Number = ((0x100 | Mantissa) « exp) » 7; else Number = mantissa;

DWord	Bit	Description
0	31:24	**Tearing_Count 1 (FMD Variance[8])** Format: U8 Number of pixels that have (diff_cTcB > diff_cTcT + diff_cBcB) Value 0 — Name — Description: DI is Disabled
	23:16	**Tearing_Count 2** Format: U8 If the frame is Deinterlaced with Top First in the DN/DI state then this is (FMD Variance[9]) = Number of pixels that have (diff_cTpB > diff_cTcT + diff_pBpB) If the frame is bottom first then this is (FMD Variance[10]) = Number of pixels that have (diff_cBpT > diff_pTpT + diff_cBcB) Value 0 — Name — Description: DI is Disabled
	15:8	**Motion_Count (FMD Variance[7])** Format: U8 Number of pixels that are moving (different above a threshold) Value 0 — Name — Description: DI is Disabled
	7:0	**Reserved** Format: MBZ
1	31:28	**sSTAD** Format: U4 Shift for the Sum in time of absolute differences for 16x4. Value 0 — Name — Description: DN is Disabled — Project: CHV, BSW

Encoder Statistics Format

	27:24	**sSHCM** Format: U4 Shift for the Sum horizontaly of absolute differences. Value 0 — Name — Description: DN is Disabled
	23:20	**sSVCM** Format: U4 Shift for the Sum vertically of absolute differences.
	19:16	**sDiff_cTpT** Format: U4 Shift for the sum of differences in top fields of current and previous frame. Value 0 — Name — Description: DI is Disabled
	15:12	**sDiff_cBpB** Format: U4 Shift for the sum of differences in bottom field of current and previous frame. Value 0 — Name — Description: DI is Disabled
	11:8	**sDiff_cTcB** Format: U4 Shift for the sum of differences between top and bottom field in current frame. Value 0 — Name — Description: DI is Disabled
	7:4	**sDiff_cTpB** Format: U4 Shift for the sum of differences between current top and previous bottom. Value 0 — Name — Description: DI is Disabled
	3:0	**sDiff_cBpT** Format: U4 Shift for the sum of differences between current bottom and previous top. Value 0 — Name — Description: DI is Disabled
2	31:24	**mDiff_cBpB (FMD Variance[1])** Format: U8 Mantissa of sum of differences in bottom field of current and previous frame.

Encoder Statistics Format

		Value / **Name** / **Description**: 0 / / DI is Disabled
	23:16	**mDiff_cTcB (FMD Variance[2])** Format: U8 Mantissa of sum of differences between top and bottom field in current frame. **Value** / **Name** / **Description**: 0 / / DI is Disabled
	15:8	**mDiff_cTpB (FMD Variance[3])** Format: U8 Mantissa of sum of differences between current top and previous bottom. **Value** / **Name** / **Description**: 0 / / DI is Disabled
	7:0	**mDiff_cBpT (FMD Variance[4])** Format: U8 Mantissa of sum of differences between current bottom and previous top. **Value** / **Name** / **Description**: 0 / / DI is Disabled
3	31:24	**mSTAD** Format: U8 Mantissa of Sum in time of absolute differences for 16x4. **Value** / **Name** / **Description** / **Project**: 0 / / DN is Disabled / CHV, BSW
	23:16	**mSHCM** Format: U8 Mantissa of Sum horizontaly of absolute differences. **Value** / **Name** / **Description**: 0 / / DN is Disabled
	15:8	**mM** Format: U8 Mantissa of Sum vertically of absolute differences. **Value** / **Name** / **Description**: 0 / / DN is Disabled
	7:0	**mDiff_cTpT (FMD Variance[0])** Format: U8 Mantissa of sum of differences in top fields of current and previous frame. **Value** / **Name** / **Description**: 0 / / DI is Disabled

EU_INSTRUCTION_BASIC_ONE_SRC

EU_INSTRUCTION_BASIC_ONE_SRC		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Exists If: ([Operand Controls][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([Operand Controls][Src0.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_IMM32 [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_BASIC_THREE_SRC

EU_INSTRUCTION_BASIC_THREE_SRC

Project: CHV, BSW
Source: Eulsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127	**Reserved** Format: MBZ
	126:106	**Source 2** Project: CHV, BSW Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	105:85	**Source 1** Project: CHV, BSW Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	84:64	**Source 0** Project: CHV, BSW Format: EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC [CHV, BSW]
	63:56	**Destination Register Number** Format: DstRegNum [CHV, BSW]
	55:53	**Destination Subregister Number**
	52:49	**Destination Channel Enable** Format: ChanEn[4] Four channel enables are defined for controlling which channels are written into the destination region. These channel mask bits are applied in a modulo-four manner to all ExecSize channels. There is 1-bit Channel Enable for each channel within the group of 4. If the bit is cleared, the write for the corresponding channel is disabled. If the bit is set, the write is enabled. Mnemonics for the bit being set for the group of 4 are x, y, z, and w, respectively, where x corresponds to Channel 0 in the group and w corresponds to channel 3 in the group
	48:46	**Destination Data Type** Value / Name / Description: 000b / :f / single precision Float (32-bit) 001b / :d / signed Doubleword integer 010b / :ud / Unsigned Doubleword integer 011b / :df / Double precision Float (64-bit) 100b / :hf / Half Float (16-bit)

EU_INSTRUCTION_BASIC_THREE_SRC

101b-111b	Reserved	

45:43 Source Data Type

Value	Name	Description
000b	:f	single precision Float (32-bit)
001b	:d	signed Doubleword integer
010b	:ud	Unsigned Doubleword integer
011b	:df	Double precision Float (64-bit)
100b	:hf	Half Float (16-bit)
101b-111b	Reserved	

42:41 Source 2 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	SrcMod [CHV, BSW]

40:39 Source 1 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	SrcMod [CHV, BSW]

42:37 Reserved

Exists If:	(Property[Source Modifier]=='false')
Format:	MBZ

38:37 Source 0 Modifier

Exists If:	(Property[Source Modifier]=='true')
Format:	SrcMod [CHV, BSW]

36:35 Reserved

Project:	CHV, BSW
Format:	MBZ

34 MaskCtrl

Project:	CHV, BSW

(formerly WECtrl/Write Enable Control). This flag disables the normal write enables; it should normally be 0.

Value	Name	Description
0	Normal	Use the normal write enables in Dst.ChanEn (normal setting).
1	NoMask	Write all channels except those disabled by predication or by other masks besides the write enables.

Programming Notes
MaskCtrl = NoMask also skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.

33 Flag Register Number

EU_INSTRUCTION_BASIC_THREE_SRC		
		This field contains the flag register number for instructions with a non-zero Conditional Modifier.
	32	**Flag Subregister Number** This field contains the flag subregister number for instructions with a non-zero Conditional Modifier.
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_BASIC_TWO_SRC

EU_INSTRUCTION_BASIC_TWO_SRC		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Exists If: ([RegSource][Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**ImmSource** Exists If: ([ImmSource][Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:32	**Operand Controls** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_BRANCH_CONDITIONAL

EU_INSTRUCTION_BRANCH_CONDITIONAL

Project: CHV, BSW
Source: Eulsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127:64	**Sources** Exists If: ([Src1.RegFile]!='IMM') Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	127:64	**Sources** Exists If: ([Src1.RegFile]=='IMM') Format: EU_INSTRUCTION_SOURCES_REG_IMM [CHV, BSW]
	63:48	**JIP** Format: S15 Jump Target Offset. The jump distance in number of eight-byte units if a jump is taken for the instruction.
	47	**Reserved** Format: MBZ
	46:44	**Src1.SrcType** Format: DataType This field specifies the numeric data type of the source operand src1. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on RegFile field of the source operand, there are two different encoding for this field. If a source is a register operand, this field follows the Source Register Type Encoding. If a source is an immediate operand, this field follows the Source Immediate Type Encoding. **Programming Notes** Both source operands, src0 and src1, support immediate types, but only one immediate is allowed for a given instruction and it must be the last operand. Halfbyte integer vector (v) type can only be used in instructions in packed-word execution mode. Therefore, in a two-source instruction where src1 is of type :v, src0 must be of type :b, :ub, :w, or :uw.
	43:42	**Src1.RegFile** Format: RegFile [CHV, BSW]
	41:39	**Src0.SrcType** Format: DataType

EU_INSTRUCTION_BRANCH_CONDITIONAL

	38:37	**Src0.RegFile** Format: RegFile [CHV, BSW]
	36:34	**Destination Data Type** Format: DataType This field specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send instruction, this field applies to the CurrDst ? the current destination operand.
	33:32	**Destination Register File** Format: RegFile [CHV, BSW] **Value** / **Name** / **Description** 11b / Reserved / Note that it is obvious that immediate cannot be a destination operand.
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_BRANCH_ONE_SRC

EU_INSTRUCTION_BRANCH_ONE_SRC

Project: CHV, BSW
Source: Eulsa
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 Jump Target Offset. The relative offset in bytes if a jump is taken for the instruction.
	95	**Source 0 Address Immediate [9] Sign Bit** Project: CHV, BSW
	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW]
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_BRANCH_TWO_SRC

EU_INSTRUCTION_BRANCH_TWO_SRC		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:96	**JIP** Project: CHV, BSW Format: S31 The byte-aligned jump distance if a jump is taken for the channel.
	95:64	**UIP** Project: CHV, BSW Format: S31 The byte aligned jump distance if a jump is taken for the instruction.
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:0	**Header** Format: EU_INSTRUCTION_HEADER [CHV, BSW]

EU_INSTRUCTION_COMPACT_THREE_SRC

EU_INSTRUCTION_COMPACT_THREE_SRC

Project: CHV, BSW
Source: Eulsa
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:57	**Src2.RegNum[6:0]** Format: SrcRegNum[6:0] Src2.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src2.RegNum[7]. Maps to 124:118
	56:50	**Src1.RegNum[6:0]** Format: SrcRegNum[6:0] Src1.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src1.RegNum[7]. Maps to 103:97
	49:43	**Src0.RegNum[6:0]** Format: SrcRegNum[6:0] Src0.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src0.RegNum[7]. Maps to 82:76
	42:40	**Src2.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 117:115
	39:37	**Src1.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 96:94
	36:34	**Src0.SubRegNum** Format: SrcSubRegNum[4:2] Maps to 75:73
	33	**Src2.RepCtrl**

EU_INSTRUCTION_COMPACT_THREE_SRC

		Format: RepCtrl [CHV, BSW] Maps to 106
	32	**Src1.RepCtrl** Format: RepCtrl [CHV, BSW] Maps to 85
	31	**Reserved** Exists If: (Property[Saturation]=='false') Format: MBZ
	31	**Saturate** Exists If: (Property[Saturation]=='true') Maps to 31
	30	**Reserved**
	29	**Compaction Control** Format: CmptCtrl
	28	**Src0.RepCtrl** Format: RepCtrl [CHV, BSW] Maps to 64
	27:19	**Reserved** Format: MBZ
	18:12	**Dst.RegNum[6:0]** Format: DstRegNum[6:0] Dst.RegNum[7:0] with MSB of zero and [6:0] from the compact instruction Maps to 63:56 (Dst.RegNum)
	11:10	**SourceIndex** Project: CHV, BSW Lookup one of four 49-bit values. That value is used (from MSB to LSB) for the Src2.SubRegNum, Src2.RegNum[7], Src1.RegNum, Src1.RegNum[7], Src0.SubRegNum, Src0.RegNum[7], Src2.ChanSel, Src1.ChanSel, Src0.ChanSel, Dst.SubRegNum, Dst.ChanEnable, Dst.DstType, SrcType, Src2.Modifier, Src1.Modifier, and Src0.Modifier bit fields Maps to 126, 125, 105, 104, 84, 83, 114:107, 93:86, 72:65, 55:49, 48:43, 42:37

Value	Name	Description

EU_INSTRUCTION_COMPACT_THREE_SRC

0	00000011100100111001001110010000011110000000000000	No Negation
1	00000011100100111001001110010000011110000000000010	Negate Src0
2	00000011100100111001001110010000011110000000001000	Negate Src1
3	00000011100100111001001110010000011110000000100000	Negate Src2

9:8 **ControlIndex**

Project:	CHV, BSW

Lookup one of four 26-bit values. That value is used (from MSB to LSB) for the Src1.Type, Src2.Type MaskCtrl, FlagRegNum/FlagSubRegNum, AccWrCtrl, CondModifier, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, NibCtrl, DepCtrl, and AccessMode bit fields.

Maps to 36:35, 34, 33:32, 28:8

Value	Name	Description
0	00100000000110000000000001	(8) Q1 NoMask Align16
1	00000000000110000000000001	(8) Q1 Align16
2	00000000001000000000000001	(16) H1 Align16
3	00000000001000000000100001	(16) H2 Align16

7 **Reserved**

Format:	MBZ

6:0 **Opcode**

EU_INSTRUCTION_COMPACT_TWO_SRC

EU_INSTRUCTION_COMPACT_TWO_SRC

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

The following table describes the EU compact instruction format. The compact instruction format for 1 or 2-source instructions is essentially identical to the compact instruction format for earlier generations, but the compact fields expand to somewhat different fields in the native instruction format, as the native instruction format changed for CHV, BSW.

DWord	Bit	Description
0..1	63:56	**Src1.RegNum** Exists If: ([DataTypeIndex][Src1.RegFile]!='IMM') Format: SrcRegNum [CHV, BSW] Maps to 108:101 (Src1.RegNum)
	63:56	**Src1.RegNum** Exists If: ([DataTypeIndex][Src1.RegFile]=='IMM') Maps to 103:96 (Imm32[7:0])
	55:48	**Src0.RegNum** Format: SrcRegNum [CHV, BSW] Maps to 76:69 (Src0.RegNum)
	47:40	**Dst.RegNum** Format: DstRegNum [CHV, BSW] Maps to 60:53 (Dst.RegNum)
	39:35	**Src1Index** Exists If: ([DataTypeIndex][Src1.RegFile]!='IMM') Format: SrcIndex [CHV, BSW] If not an immediate operand, lookup one of 32 12-bit values that maps to bits 120:109. That value is used (from MSB to LSB) for the Src1.VertStride, various Src1 bit fields based on AccessMode (Src1.ChanSel[7:4], Src1.Width, Src1.HorzStride), Src1.AddrMode, and Src1.SrcMod bit fields Maps to 120:109
	39:35	**Src1Index** Exists If: ([DataTypeIndex][Src1.RegFile]=='IMM')

EU_INSTRUCTION_COMPACT_TWO_SRC

If an immediate operand, there is no lookup. Determines bits 127:104 (Imm32[31:8]) as follows: map bits 39:35 directly to bits 108:104. Sign extend to fill bits 127:109. Compact format bit 39 is thus copied to all of bits 127:108 for an immediate operand.

Maps to 127:104

34:30 **Src0Index**

Format:	SrcIndex [CHV, BSW]

Lookup one of 32 12-bit values. That value is used (from MSB to LSB) for the Src0.VertStride, various Src0 bit fields based on AccessMode (Src0.ChanSel[7:4], Src0.Width, Src0.HorzStride), Src0.AddrMode, and Src0.SrcMod bit fields. Note that this field spans a DWord boundary within the QWord compacted instruction.

Maps to 88:77

29 **Compaction Control**

Format:	CmptCtrl

28 **Reserved**

Format:	MBZ

27:24 **Reserved**

Exists If:	(Property[Conditional Modifier]=='false')
Format:	MBZ

27:24 **Conditional Modifier**

Exists If:	(Property[Conditional Modifier]=='true')
Format:	CondModifier [CHV, BSW]

23 **Accumulator Write Control**

Format:	AccWrCtrl

22:18 **SubRegIndex**

Lookup one of 32 15-bit values. That value is used (from MSB to LSB) for various fields for Src1, Src0, and Dst, including ChanEn/ChanSel, SubRegNum, and AddrImm[4] or AddrImm[4:0], depending on AddrMode and AccessMode.

Maps to 100:96, 68:64, 52:48

Value	Name	Description
0	000000000000000	0 | 0 | 0 |
1	000000000000001	0.x | 0.xx | 0.xx
2	000000000001000	8 | 0 | 0 |
3	000000000001111	0.xyzw | 0.xx | 0.xx
4	000000000010000	16 | 0 | 0 |
5	000000010000000	0 | 4 | 0 |

EU_INSTRUCTION_COMPACT_TWO_SRC

		6	000000100000000	0 | 8 | 0 |
		7	000000110000000	0 | 12 | 0 |
		8	000001000000000	0 | 16 | 0 |
		9	000001000010000	16 | 16 | 0 |
		10	000001010000000	0 | 20 | 0 |
		11	001000000000000	0 | 0 | 4 |
		12	001000000000001	0.x | 0.xx | 0.xy
		13	001000010000001	0.x | 0.xy | 0.xy
		14	001000010000010	0.y | 0.xy | 0.xy
		15	001000010000011	0.xy | 0.xy | 0.xy
		16	001000010000100	0.z | 0.xy | 0.xy
		17	001000010000111	0.xyz | 0.xy | 0.xy
		18	001000010001000	0.w | 0.xy | 0.xy
		19	001000010001110	0.yzw | 0.xy | 0.xy
		20	001000010001111	0.xyzw | 0.xy | 0.xy
		21	001000110000000	0 | 12 | 4 |
		22	001000111101000	0.w | 0.ww | 0.xy
		23	010000000000000	0 | 0 | 8 |
		24	010000110000000	0 | 12 | 8 |
		25	011000000000000	0 | 0 | 12 |
		26	011110010000111	0.xyz | 0.xy | 0.ww
		27	100000000000000	0 | 0 | 16 |
		28	101000000000000	0 | 0 | 20 |
		29	110000000000000	0 | 0 | 24 |
		30	111000000000000	0 | 0 | 28 |
		31	111000000011100	28 | 0 | 28 |

	17:13 **DataTypeIndex**
	Lookup one of 32 21-bit values. That value is used (from MSB to LSB) for the Dst.AddrMode, Dst.HorzStride, Src1.SrcType, Src1.RegFile, Src0.SrcType, Src0.RegFile, Dst.DstType, and Dst.RegFile bit fields.
	Maps to 63:61, 94:89, 46:35

Value	Name	Description
0	00100000000000000001	r:ud | a:ud | a:ud | <1> | dir |
1	00100000000001000000	a:ud | r:ud | a:ud | <1> | dir |
2	00100000000001000001	r:ud | r:ud | a:ud | <1> | dir |
3	00100000000011000001	r:ud | i:ud | a:ud | <1> | dir |

EU_INSTRUCTION_COMPACT_TWO_SRC

		4	00100000000101011101	r:f | r:d | a:ud | <1> | dir |
		5	00100000010111011101	r:f | i:vf | a:ud | <1> | dir |
		6	00100000011101000001	r:ud | r:f | a:ud | <1> | dir |
		7	00100000011101000101	r:d | r:f | a:ud | <1> | dir |
		8	00100000011101011101	r:f | r:f | a:ud | <1> | dir |
		9	00100001000001000001	r:ud | r:ud | r:ud | <1> | dir |
		10	00100001100000010000000	a:ud | r:ud | i:ud | <1> | dir |
		11	00100001100000010000001	r:ud | r:ud | i:ud | <1> | dir |
		12	00100010100010100101	r:d | r:d | r:d | <1> | dir |
		13	00100011100010100100	a:d | r:d | i:d | <1> | dir |
		14	00100011100010100101	r:d | r:d | i:d | <1> | dir |
		15	00101110001110101101	r:f | r:f | a:f | <1> | dir |
		16	00101110101110001101	r:f | a:f | r:f | <1> | dir |
		17	00101110101110101100	a:f | r:f | r:f | <1> | dir |
		18	00101110101110101101	r:f | r:f | r:f | <1> | dir |
		19	00101111101110101100	a:f | r:f | i:f | <1> | dir |
		20	00000000010000001100	a:w | a:ub | a:ud | <0> | dir |
		21	00100000000001011101	r:f | r:ud | a:ud | <1> | dir |
		22	00100000000101000101	r:d | r:d | a:ud | <1> | dir |
		23	00100001000001000000	a:ud | r:ud | r:ud | <1> | dir |
		24	00100010100010100100	a:d | r:d | r:d | <1> | dir |
		25	00100011100010000100	a:d | a:d | i:d | <1> | dir |
		26	00100100100100001001	r:uw | a:uw | r:uw | <1> | dir |
		27	00101011101110101101	r:f | r:f | i:vf | <1> | dir |
		28	00101111101110101101	r:f | r:f | i:f | <1> | dir |
		29	00100111100110100100	a:w | r:w | i:w | <1> | dir |
		30	00100100100100100100	a:uw | r:uw | r:uw | <1> | dir |
		31	00100101100100100100	a:uw | r:uw | i:uw | <1> | dir |
	12:8	**ControlIndex**		
		Lookup one of 32 19-bit values. That value is used (from MSB to LSB) for the FlagRegNum, FlagSubRegNum, Saturate, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, DepCtrl, MaskCtrl, and AccessMode bit fields.		
		Maps to 33:32, 31, 23:12, 10:9, 34, 8		

Value	Name	Description
0	0000000000000000010	Align1 | We | (1) | f0.0
1	0000100000000000000	Align1 | (4) | f0.0

EU_INSTRUCTION_COMPACT_TWO_SRC

2	0000100000000000001	Align16 | (4) | f0.0
3	0000100000000000010	Align1 | We | (4) | f0.0
4	0000100000000000011	Align16 | We | (4) | f0.0
5	0000100000000000100	Align1 | NoDDClr | (4) | f0.0
6	0000100000000000101	Align16 | NoDDClr | (4) | f0.0
7	0000100000000000111	Align16 | We | NoDDClr | (4) | f0.0
8	0000100000000001000	Align1 | NoDDChk | (4) | f0.0
9	0000100000000001001	Align16 | NoDDChk | (4) | f0.0
10	0000100000000001101	Align16 | NoDDClr, NoDDChk | (4) | f0.0
11	0000110000000000000	Align1 | Q1 | (8) | f0.0
12	0000110000000000001	Align16 | Q1 | (8) | f0.0
13	0000110000000000010	Align1 | We | Q1 | (8) | f0.0
14	0000110000000000011	Align16 | We | Q1 | (8) | f0.0
15	0000110000000000100	Align1 | NoDDClr | Q1 | (8) | f0.0
16	0000110000000000101	Align16 | NoDDClr | Q1 | (8) | f0.0
17	0000110000000000111	Align16 | We | NoDDClr | Q1 | (8) | f0.0
18	0000110000000001001	Align16 | NoDDChk | Q1 | (8) | f0.0
19	0000110000000001101	Align16 | NoDDClr, NoDDChk | Q1 | (8) | f0.0
20	0000110000000010000	Align1 | Q2 | (8) | f0.0
21	0000110000100000000	Align1 | Q1 | +f.xyzw | (8) | f0.0
22	0001000000000000000	Align1 | H1 | (16) | f0.0
23	0001000000000000010	Align1 | We | H1 | (16) | f0.0
24	0001000000000000100	Align1 | NoDDClr | H1 | (16) | f0.0
25	0001000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.0
26	0010110000000000000	Align1 | Q1 | (8) | .sat | f0.0
27	0010110000000010000	Align1 | Q2 | (8) | .sat | f0.0
28	0011000000000000000	Align1 | H1 | (16) | .sat | f0.0
29	0011000000100000000	Align1 | H1 | +f.xyzw | (16) | .sat | f0.0
30	0101000000000000000	Align1 | H1 | (16) | f0.1
31	0101000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.1

7	**Reserved**
6:0	**Opcode**

EU_INSTRUCTION_CONTROLS_A

EU_INSTRUCTION_CONTROLS_A

Project: CHV, BSW
Source: EuIsa
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:13	**ExecSize** Format: ExecSize [CHV, BSW] This field determines the number of channels operating in parallel for this instruction. The size cannot exceed the maximum number of channels allowed for the given data type.
	12	**Reserved** Exists If: (Property[Predication]=='false')
	12	**PredInv** Exists If: (Property[Predication]=='true') This field, together with PredCtrl, enables and controls the generation of the predication mask for the instruction. When it is set, the predication uses the inverse of the predication bits generated according to setting of Predicate Control. In other words, effect of PredInv happens after PredCtrl. This field is ignored by hardware if Predicate Control is set to 0000 there is no predication. PMask is the final predication mask produced by the effects of both fields.
	11:8	**Reserved** Exists If: (Property[Predication]=='false') Format: PredCtrl [CHV, BSW]
	11:8	**PredCtrl** Exists If: (Property[Predication]=='true') Format: PredCtrl [CHV, BSW] This field, together with PredInv, enables and controls the generation of the predication mask for the instruction. It allows per-channel conditional execution of the instruction based on the content of the selected flag register. Encoding depends on the access mode. In Align16 access mode, there are eight encodings (including no predication). All encodings are based on group-of-4 predicate bits, including channel sequential, replication swizzles and horizontal any|all operations. The same configuration is repeated for each group-of-4 execution channels.

Value	Name	Description
0	Positive **[Default]**	Positive polarity of predication. Use the predication mask produced by PredCtrl
1	Negative	Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

EU_INSTRUCTION_CONTROLS_A

7:6 **Thread Control**

Format:	ThreadCtrl [CHV, BSW]

Thread Control. This field provides explicit control for thread switching. If this field is set to 00b, it is up to the GEN execution units to manage thread switching. This is the normal (and unnamed) mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. In another example, if the current instruction is ready to go, however, there is another thread with higher priority that also has an instruction ready, the EU switches to that thread. If this field is set to Switch, a forced thread switch occurs after the current instruction is executed and before the next instruction. In addition, a long delay (longer than the execution pipe latency) is introduced for the current thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. Switch is designed primarily as a safety feature in case there are race conditions for certain instructions.

5:4 **QtrCtrl**

Format:	QtrCtrl [CHV, BSW]

Quarter Control. This field provides explicit control for ARF selection. This field combined with NibCtrl and ExecSize determines which channels are used for the ARF registers.

3 **NibCtrl**

Nibble Control. This field is used in some instructions along with QtrCtrl. See the description of QtrCtrl below. NibCtrl is only used for SIMD4 instructions with a DF (Double Float) source or destination.

Value	Name	Description
0	Odd	Use an odd 1/8th for DMask/VMask and ARF (first, third, fifth, or seventh depending on QtrCtrl).
1	Even	Use an even 1/8th for DMask/VMask and ARF (second, fourth, sixth, or eighth depending on QtrCtrl).

Programming Notes
Note that if eighths are given zero-based indices from 0 to 7, then NibCtrl = 0 indicates even indices and NibCtrl = 1 indicates odd indices.

2:1 **DepCtrl**

Format:	DepCtrl [CHV, BSW]

Destination Dependency Control. This field selectively disables destination dependency check and clear for this instruction. When it is set to 00, normal destination dependency control is performed for the instruction hardware checks for destination hazards to ensure data integrity. Specifically, destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard will be cleared when the destination operands retire. When bit 10 is set (NoDDClr), the destination register scoreboard will NOT be cleared when the destination operands retire. When bit 11 is set (NoDDChk), hardware does not check for destination register dependency before the instruction is made ready for execution. NoDDClr and NoDDChk are not mutual exclusive. When this field is

EU_INSTRUCTION_CONTROLS_A

		not all-zero, hardware does not protect against destination hazards for the instruction. This is typically used to assemble data in a fine grained fashion (e.g. matrix-vector compute with dot-product instructions), where the data integrity is guaranteed by software based on the intended usage of instruction sequences.
	0	**AccessMode** Access Mode. This field determines the operand access for the instruction. It applies to all source and destination operands. When it is cleared (Align1), the instruction uses byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are not supported. When it is set (Align16), the instruction uses 16-byte-aligned addressing for all source and destination operands. Source swizzle control and destination mask control are supported in this mode.

Value	Name
0	Align1 **[Default]**
1	Align16

EU_INSTRUCTION_CONTROLS_B

EU_INSTRUCTION_CONTROLS_B

Project: CHV, BSW
Source: Eulsa
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3	**Reserved** Exists If: (Property[Saturation]=='false') Format: MBZ
	3	**Saturate** Exists If: (Property[Saturation]=='true') Enables or disables destination saturation. When it is set, output values to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any value outside the saturation target range for the data type to the closest value in the target range. For a floating-point destination type, the saturation target range is [0.0, 1.0]. For a floating-point NaN, there is no closest value; any NaN saturates to 0.0. Note that enabling Saturate overrides all of the NaN propagation behaviors described for various numeric instructions. Any floating-point number greater than 1.0, including +INF, saturates to 1.0. Any negative floating-point number, including -INF, saturates to 0.0. Any floating-point number in the range 0.0 to 1.0 is not changed by saturation. For an integer destination type, the maximum range for that type is the saturation target range. For example, the saturation range for B (Signed Byte Integer) is [-128, 127]. When Saturate is clear, destination values are not saturated. For example, a wrapped result (modulo) is output to the destination for an overflowed integer value. See the Numeric Data Typessection for information about data types and their ranges. Value / Name / Description: 0 — No destination modification **[Default]** 1 — sat — Saturate the output
	2	**Reserved**
	1	**CmptCtrl** Compaction Control Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format [CHV, BSW] for more information. Value / Name / Description: 0 — NoCompaction — No compaction. 128-bit native instruction supporting all instruction options. 1 — Compacted — Compaction is enabled. 64-bit compact instruction supporting only some

EU_INSTRUCTION_CONTROLS_B

		instruction variations.
0	**AccWrCtrl** AccWrCtrl. This field allows per instruction accumulator write control.	

Value	Name	Description
0	Don't write to ACC **[Default]**	
1	Update ACC	Write result to the ACC, and destination

EU_INSTRUCTION_CONTROLS

EU_INSTRUCTION_CONTROLS		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	24	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	23:20	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B [CHV, BSW]
	19:16	**Reserved** Exists If: (Property[Conditional Modifier]=='false') Format: MBZ
	19:16	**CondModifier** Exists If: (Property[Conditional Modifier]=='true') Format: CondModifier [CHV, BSW] Does not exist for send/sendc/math/branch/break-continue opcodes
	15:0	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A [CHV, BSW]

EU_INSTRUCTION_HEADER

EU_INSTRUCTION_HEADER		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:8	**Control** Format: EU_INSTRUCTION_CONTROLS [CHV, BSW]
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

EU_INSTRUCTION_ILLEGAL

EU_INSTRUCTION_ILLEGAL		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

EU_INSTRUCTION_MATH

EU_INSTRUCTION_MATH		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:64	**RegSource** Format: EU_INSTRUCTION_SOURCES_REG_REG [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B [CHV, BSW]
	27:24	**Function Control (FC)** Format: FC [CHV, BSW]
	23:8	**Controls A** Format: EU_INSTRUCTION_CONTROLS_A [CHV, BSW]
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

EU_INSTRUCTION_NOP

EU_INSTRUCTION_NOP		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	127:31	**Reserved** Format: MBZ
	30	**Reserved**
	29:7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

EU_INSTRUCTION_OPERAND_CONTROLS

EU_INSTRUCTION_OPERAND_CONTROLS

Project: CHV, BSW
Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:16	**Destination Register Region** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_DST_ALIGN16 [CHV, BSW]
	31:16	**Destination Register Region** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_DST_ALIGN1 [CHV, BSW]
	15	**Reserved** Exists If: ([Destination Register Region][Destination Addressing Mode]=='Direct') Format: MBZ
	15	**Destination Address Immediate[9:9]** Exists If: ([Destination Register Region][Destination Addressing Mode]=='Indirect') Format: U1
	14:11	**Src0.SrcType** Exists If: ([Src0.RegFile]!='IMM') Format: SrcType [CHV, BSW]
	14:11	**Src0.SrcType** Exists If: ([Src0.RegFile]=='IMM') Format: SrcImmType [CHV, BSW]
	10:9	**Src0.RegFile** Format: RegFile [CHV, BSW]
	8:5	**Destination Data Type** Format: DstType [CHV, BSW] This field specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a send instruction, this field applies to the CurrDst the current destination operand.
	4:3	**Destination Register File** Format: RegFile [CHV, BSW]

EU_INSTRUCTION_OPERAND_CONTROLS

		Value / **Name** / **Description** 11b / Reserved / Note that it is obvious that immediate cannot be a destination operand.
	2	**MaskCtrl** Mask Control (formerly Write Enable Control). This field determines if the the per channel write enables are used to generate the final write enable. This field should be normally "0". **Value** / **Name** / **Description** 0 / Normal **[Default]** / 1 / Write all channels / Except channels killed with predication control **Programming Notes** MaskCtrl = NoMask skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.
	1:0	**Flag Register Number/Subregister Number**

EU_INSTRUCTION_OPERAND_DST_ALIGN1

EU_INSTRUCTION_OPERAND_DST_ALIGN1		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15	**Destination Addressing Mode** Format: AddrMode [CHV, BSW] For a send instruction, this field applies to PostDst - the post destination operand. Addressing mode for CurrDst (current destination operand) is fixed as Direct. (See Instruction Reference chapter for CurrDst and PostDst.)
	14:13	**Destination Horizontal Stride** Format: HorzStride [CHV, BSW] For a send instruction, this field applies to CurrDst. PostDst only uses the register number.
	12:9	**Destination Address Subregister Number** Project: CHV, BSW Exists If: ([Destination Addressing Mode]=='Indirect') Format: AddrSubRegNum [CHV, BSW] For a send instruction, this field applies to PostDst
	12:5	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: DstRegNum [CHV, BSW] For a send instruction, this field applies to PostDst.
	8:0	**Destination Address Immediate** Project: CHV, BSW Exists If: ([Destination Addressing Mode]=='Indirect') Format: S8 For a send instruction, this field applies to PostDst.
	4:0	**Destination Subregister Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: DstSubRegNum [CHV, BSW] For a send instruction, this field applies to CurrDst.

EU_INSTRUCTION_OPERAND_DST_ALIGN1		

EU_INSTRUCTION_OPERAND_DST_ALIGN16

EU_INSTRUCTION_OPERAND_DST_ALIGN16

Project: CHV, BSW
Source: Eulsa
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15	**Destination Addressing Mode** Format: AddrMode [CHV, BSW] For a send instruction, this field applies to PostDst - the post destination operand. Addressing mode for CurrDst (current destination operand) is fixed as Direct. (See Instruction Reference chapter for CurrDst and PostDst.)
	14:13	**Reserved** Value: 01b — Name: See Programming Note **Programming Notes** Although Dst.HorzStride is a don?t care for Align16, HW needs this to be programmed as ?01?.
	12:9	**Destination Address Subregister Number** Project: CHV, BSW Exists If: ([Destination Addressing Mode]=='Indirect') Format: AddrSubRegNum [CHV, BSW] For a send instruction, this field applies to PostDst
	12:5	**Destination Register Number** Exists If: ([Destination Addressing Mode]=='Direct') Format: DstRegNum [CHV, BSW] For a send instruction, this field applies to PostDst.
	8:4	**Destination Address Immediate[8:4]** Project: CHV, BSW Exists If: ([Destination Addressing Mode]=='Indirect') Format: S8[8:4] For a send instruction, this field applies to PostDst
	4	**Destination Subregister Number** Exists If: ([Destination Addressing Mode]=='Direct')

EU_INSTRUCTION_OPERAND_DST_ALIGN16		
		Format: DstSubRegNum[4:4] For a send instruction, this field applies to CurrDst.
	3:0	**Destination Channel Enable** Format: ChanEn[4] For a send instruction, this field applies to the CurrDst

EU_INSTRUCTION_OPERAND_SEND_MSG

EU_INSTRUCTION_OPERAND_SEND_MSG

Project: CHV, BSW
Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**EOT** **Description** / **Project**: This field controls the termination of the thread. For a send instruction, if this field is set, EU will terminate the thread and also set the EOT bit in the message sideband. This field only applies to the send instruction. It is not present for other instructions. — CHV, BSW **Value** / **Name**: 0 — Thread is not terminated; 1 — EOT

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	25
Default Value:	0x00000000

DWord	Bit	Description
0	24:21	**Source Vertical Stride** Format: VertStride [CHV, BSW]
	20:18	**Source Width** Format: Width [CHV, BSW]
	17:16	**Source Horizontal Stride** Format: HorzStride [CHV, BSW]
	15	**Source Addressing Mode** Format: AddrMode [CHV, BSW]
	14:13	**Reserved** Exists If: (Property[Source Modifier]=='false') Format: MBZ
	14:13	**Source Modifier** Exists If: (Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	12:9	**Source Address Subregister Number** Project: CHV, BSW Exists If: ([Source Addressing Mode]=='Indirect') Format: AddrSubRegNum [CHV, BSW]
	12:5	**Source Register Number** Exists If: ([Source Addressing Mode]=='Direct') Format: SrcRegNum [CHV, BSW]
	8:0	**Source Address Immediate [8:0]** Project: CHV, BSW Exists If: ([Source Addressing Mode]=='Indirect') Format: S9[8:0]
	4:0	**Source Subregister Number** Exists If: ([Source Addressing Mode]=='Direct') Format: SrcSubRegNum [CHV, BSW]

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16

Project: CHV, BSW
Source: Eulsa
Size (in bits): 25
Default Value: 0x00000000

DWord	Bit	Description
0	24:21	**Source Vertical Stride** Format: VertStride [CHV, BSW]
	20	**Reserved** Format: MBZ
	19:16	**Source Channel Select[7:4]** Format: ChanSel[4][7:4]
	15	**Source Addressing Mode** Format: AddrMode [CHV, BSW]
	14:13	**Reserved** Exists If: (Property[Source Modifier]=='false') Format: MBZ
	14:13	**Source Modifier** Exists If: (Property[Source Modifier]=='true') Format: SrcMod [CHV, BSW]
	12:9	**Source Address Subregister Number** Project: CHV, BSW Exists If: ([Source Addressing Mode]=='Indirect') Format: AddrSubRegNum [CHV, BSW]
	12:5	**Source Register Number** Exists If: ([Source Addressing Mode]=='Direct') Format: SrcRegNum [CHV, BSW]
	8:4	**Source Address Immediate[8:4]** Project: CHV, BSW Exists If: ([Source Addressing Mode]=='Indirect') Format: S9[8:4]
	4	**Source Subregister Number[4:4]** Exists If: ([Source Addressing Mode]=='Direct') Format: SrcSubRegNum[4:4]

EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16			
	3:0	**Source Channel Select[3:0]**	
		Format:	ChanSel[4][3:0]

EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC

EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	21	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	20	**Source Subregister Number [1]** Project: CHV, BSW Format: SrcSubRegNum[1]
	19:12	**Source Register Number** Format: SrcRegNum [CHV, BSW]
	11:9	**Source Subregister Number [4:2]** Format: SrcSubRegNum[4:2]
	8:1	**Source Swizzle** Format: ChanSel[4]
	0	**Source Replicate Control** Format: RepCtrl [CHV, BSW]

EU_INSTRUCTION_SEND

EU_INSTRUCTION_SEND

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0..3	127:96	**Message** Format: EU_INSTRUCTION_OPERAND_SEND_MSG [CHV, BSW]
	95	**Reserved** Format: MBZ
	94:91	**Src1.SrcType** Project: CHV, BSW Format: SrcType [CHV, BSW] Value: 11b — Name: Reserved
	90:89	**Src1.RegFile** Project: CHV, BSW Format: RegFile [CHV, BSW]
	88:64	**Source 0** Project: CHV, BSW Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	88:64	**Source 0** Project: CHV, BSW Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	63:32	**Operand Control** Format: EU_INSTRUCTION_OPERAND_CONTROLS [CHV, BSW]
	31:28	**Controls B** Format: EU_INSTRUCTION_CONTROLS_B [CHV, BSW]
	27:24	**Shared Function ID (SFID)** Format: SFID [CHV, BSW]
	23:8	**Controls A**

EU_INSTRUCTION_SEND

		Format: EU_INSTRUCTION_CONTROLS_A [CHV, BSW]
	7	**Reserved** Format: MBZ
	6:0	**Opcode** Format: EU_OPCODE [CHV, BSW]

EU_INSTRUCTION_SOURCES_IMM32

EU_INSTRUCTION_SOURCES_IMM32		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Single source, immediate		
DWord	**Bit**	**Description**
0..1	63:32	**Source 0 Immediate**
	31:25	**Reserved** Format: MBZ
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]

EU_INSTRUCTION_SOURCES_REG

EU_INSTRUCTION_SOURCES_REG		
Project:	CHV, BSW	
Source:	Eulsa	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Single source, register		
DWord	**Bit**	**Description**
0..1	63:25	**Reserved** Format: MBZ
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]

EU_INSTRUCTION_SOURCES_REG_IMM

EU_INSTRUCTION_SOURCES_REG_IMM

Project: CHV, BSW
Source: Eulsa
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

Dual source, register and immediate

DWord	Bit	Description
0..1	63:32	**Source 1 Immediate**
	31	**Reserved** Exists If: ([Source 0][Source Addressing Mode]=='Direct') Format: MBZ
	31	**Source 0 Address Immediate [9] (Sign Bit)** Exists If: ([Source 0][Source Addressing Mode]=='Indirect') Format: S9[9]
	30:27	**Src1.SrcType** Format: SrcImmType [CHV, BSW]
	26:25	**Src1.RegFile** Format: RegFile [CHV, BSW] Value / Name: 00b — Reserved 01b — Reserved
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]

EU_INSTRUCTION_SOURCES_REG_REG

EU_INSTRUCTION_SOURCES_REG_REG

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

Dual source, both registers

DWord	Bit	Description
0..1	63:58	**Reserved** Format: MBZ
	57	**Reserved** Exists If: ([Source 1][Source Addressing Mode]=='Direct') Format: MBZ
	57	**Source 1 Address Immediate [9] (Sign Bit)** Exists If: ([Source 1][Source Addressing Mode]=='Indirect') Format: S9[9]
	56:32	**Source 1** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	56:32	**Source 1** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]
	31	**Reserved** Exists If: ([Source 0][Source Addressing Mode]=='Direct') Format: MBZ
	31	**Source 0 Address Immediate [9] (Sign Bit)** Exists If: ([Source 0][Source Addressing Mode]=='Indirect') Format: S9[9]
	30:27	**Src1.SrcType** Format: SrcType [CHV, BSW] This field specifies the numeric data type of the source operand src1. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on RegFile field of the source operand, there are two different encoding for this field. If a source is a register operand, this field follows the Source Register Type Encoding. If a source is an immediate operand, this field follows the Source Immediate Type Encoding. Value / Name

EU_INSTRUCTION_SOURCES_REG_REG

		11b: Reserved
		Programming Notes Both source operands, src0 and src1, support immediate types, but only one immediate is allowed for a given instruction and it must be the last operand. Halfbyte integer vector (v) type can only be used in instructions in packed-word execution mode. Therefore, in a two-source instruction where src1 is of type :v, src0 must be of type :b, :ub, :w, or :uw.
	26:25	**Src1.RegFile** Format: RegFile [CHV, BSW]
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align16') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16 [CHV, BSW]
	24:0	**Source 0** Exists If: (Structure[EU_INSTRUCTION_CONTROLS_A][AccessMode]=='Align1') AND (Structure[EU_INSTRUCTION_OPERAND_CONTROLS][Src0.RegFile]!='IMM') Format: EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1 [CHV, BSW]

ExtMsgDescpt

ExtMsgDescpt

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0 Extended Message Descriptor Definition for SendS (Immediate)	31:16	**Extended Function Control** Project: CHV, BSW Format: U16 This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
	15:12	**Reserved** Project: CHV, BSW Format: MBZ
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10:6	**Reserved** Project: CHV, BSW Format: MBZ
	5	**EOT** Format: U1 This field, if set, indicates that this is the final message of the thread and the threads resources can be reclaimed. Value: 0 — No Termination Value: 1 — EOT
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** Format: U4 If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Value / Name

ExtMsgDescpt			
		0000b	Null
		0001b	Reserved
		0010b	SamplingEngine
		0011b	MessageGateway
		0100b	DataPortSamplerCache
		0101b	DataPortRenderCache
		0110b	URB
		0111b	ThreadSpawner
		1000b	VideoMotionEstimation
		1001b	ConstantCache
		1010b-1111b	Reserved

ExtMsgDescptImmediate

ExtMsgDescptImmediate

Project: CHV, BSW
Source: Eulsa
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0 Extended Message Descriptor Definition for SendS (Immediate)	31:16	**Extended Function Control** Format: U16 This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.
	15:12	**Reserved** Format: MBZ
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10	**Reserved** Format: MBZ
	9:6	**Reserved** Project: CHV, BSW Format: MBZ
	5	**EOT** Format: U1 This field, if set, indicates that this is the final message of the thread and the threads resources can be reclaimed. Value: 0 — Name: No Termination Value: 1 — Name: EOT
	4	**Reserved** Format: MBZ
	3:0	**Target Function ID** Format: U4 If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Value — Name

ExtMsgDescptImmediate			
		0000b	Null
		0001b	Reserved
		0010b	SamplingEngine
		0011b	MessageGateway
		0100b	DataPortSamplerCache
		0101b	DataPortRenderCache
		0110b	URB
		0111b	ThreadSpawner
		1000b	VideoMotionEstimation
		1001b	ConstantCache
		1010b-1111b	Reserved

FFTID Message Header Control

MHC_FFTID - FFTID Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:8	**Reserved** Project: All Format: Ignore Ignored
	7:0	**FFTID** Project: All Format: U8 Fixed function thread ID, used to free up resources by the thread on thread completion.

Filter_Coefficient

Filter_Coefficient		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	8	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	7:0	**Filter Coefficient** Format: S1.6 2's Complement Range : [-1 63/64, +1 63/64]

Filter_Coefficients

Filter_Coefficients		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	63:56	**Filter Coefficient Offset 7** Format: Filter_Coefficient [CHV, BSW]
	55:48	**Filter Coefficient Offset 6** Format: Filter_Coefficient [CHV, BSW]
	47:40	**Filter Coefficient Offset 5** Format: Filter_Coefficient [CHV, BSW]
	39:32	**Filter Coefficient Offset 4** Format: Filter_Coefficient [CHV, BSW]
	31:24	**Filter Coefficient Offset 3** Format: Filter_Coefficient [CHV, BSW]
	23:16	**Filter Coefficient Offset 2** Format: Filter_Coefficient [CHV, BSW]
	15:8	**Filter Coefficient Offset 1** Format: Filter_Coefficient [CHV, BSW]
	7:0	**Filter Coefficient Offset 0** Format: Filter_Coefficient [CHV, BSW]

FrameDeltaQp

FrameDeltaQp		
Source:	PRM	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:56	**FrameDeltaQp[7]** Format: S7
	55:48	**FrameDeltaQp[6]** Format: S7
	47:40	**FrameDeltaQp[5]** Format: S7
	39:32	**FrameDeltaQp[4]** Format: S7
	31:24	**FrameDeltaQp[3]** Format: S7
	23:16	**FrameDeltaQp[2]** Format: S7
	15:8	**FrameDeltaQp[1]** Format: S7
	7:0	**FrameDeltaQp[0]** Format: S7

FrameDeltaQpRange

FrameDeltaQpRange		
Source:	PRM	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:56	**FrameDeltaQpRange[7]** Format: U8
	55:48	**FrameDeltaQpRange[6]** Format: U8
	47:40	**FrameDeltaQpRange[5]** Format: U8
	39:32	**FrameDeltaQpRange[4]** Format: U8
	31:24	**FrameDeltaQpRange[3]** Format: U8
	23:16	**FrameDeltaQpRange[2]** Format: U8
	15:8	**FrameDeltaQpRange[1]** Format: U8
	7:0	**FrameDeltaQpRange[0]** Format: U8

FunctionControl

FunctionControl

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	6
Default Value:	0x00000000

DWord	Bit	Description
0	5:4	**Reserved**
	3:0	**Target Function ID**

Value	Name	Project
0000b	Reserved	
0001b	INV (Reciprocal)	
0010b	LOG	
0011b	EXP	
0100b	SQRT	
0101b	RSQ	
0110b	SIN	
0111b	COS	
1000b	Reserved	
1001b	FDIV	
1010b	POW	
1011b	INT DIV Quotient and remainder	
1100b	INT DIV Quotient only	
1101b	INT DIV Remainder only	
1110b	INVM	CHV, BSW
1111b	RSQRTM	CHV, BSW

GATHER_CONSTANT_ENTRY

GATHER_CONSTANT_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:8	**Constant Buffer Offset** Format: Offset[7:0]ConstantBuffer This field specifies the Offset in 128-bit units of the 128b entry fetched from the constant buffer for this entry (including when **On-Die Table Read Enable** is set).
	7:4	**Channel Mask** Mask: Mask[3:0] Format: ConstantBuffer Each bit of this field correspond to the 4 channels of each entry fetched from memory. When the bit is a 1, the corresponding 32-bit value is loaded in FF's push constant buffer. When the bit is a 0, the corresponding 32-bit value is not loaded. If this field is zero it means the entry is not used.
	3:0	**Binding Table Index Offset** Format: Constant Buffer Index offset [3:0]Surface State for ConstantBuffer This field specifies the Binding Table index offset from the **Constant Buffer Binding Table Block** starting point in the Binding Table. This value is added to the **Constant Buffer Binding Table Block** will result in the Binding Table Index pointing to the surface state containing the constant buffer to be referenced. If **?S Constant Buffer Dx9 Enable** is set then a value of '1' specifies that the fetch to the constant buffer should be offset by 4KB in order to address the upper 4K of the constant buffer. Any value greater than '1' is invalid when **VS Constant Buffer Dx9 Enable** is set.

Hardware-Detected Error Bit Definitions

Hardware-Detected Error Bit Definitions

Project: CHV, BSW
Source: RenderCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** Project: CHV, BSW This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: Value: 1 — Name: — Description: Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

Hardware Status Page Layout

Hardware Status Page Layout	
Project:	CHV, BSW
Source:	RenderCS
Size (in bits):	32768
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

Hardware Status Page Layout

0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Interrupt Status Register Storage** Project: All The content of the ISR register is written to this location whenever an "unmasked" bit of the ISR (as determined by the HWSTAM register) changes state.

Hardware Status Page Layout

1..3	31:0	**Reserved** Project: All Must not be used.
4	31:0	**Ring Head Pointer Storage** Project: All The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an "automatic report" (see RINGBUF registers).
5..15	31:0	**Reserved** Project: All Must not be used.
16..27	31:0	**Context Status DWords** Project: CHV, BSW
28..30 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Must not be used.
31 **Project:** CHV, BSW	31:0	**Last Written Status Offset** Project: CHV, BSW
32..39 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW
40..46	31:0	**Reserved** Project: All
47	31:0	**Reserved** Project: CHV, BSW
48..1023	31:0	**General Purpose** Project: All These locations can be used for general purpose via the MI_STORE_DATA_INDEX or MI_STORE_DATA_IMM instructions.

Header Forbidden Message Descriptor Control Field

MDC_MHF - Header Forbidden Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Project: All Format: Enumeration Indicates the message forbids a message header.

Value	Name	Description	Project
0h	No **[Default]**	Message header is not present	All
1h	Reserved	Not used	All

Header Present Message Descriptor Control Field

MDC_MHP - Header Present Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**Message Header Present** Project: All Format: Enumeration Specifies if the message uses the optional message header.

Value	Name	Description	Project
0h	No	Message header is not present	All
1h	Yes	Message header is present	All

Header Required Message Descriptor Control Field

MDC_MHR - Header Required Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1
Default Value:	0x00000001

DWord	Bit	Description
0	0	**Message Header Present** Project: All Format: Enumeration Indicates the message requires a message header.

Value	Name	Description	Project
0h	Reserved	Not used	All
1h	Yes **[Default]**	Message header is present	All

HEVC_ARBITRATION_PRIORITY

HEVC_ARBITRATION_PRIORITY

Project: CHV, BSW
Source: PRM
Size (in bits): 2
Default Value: 0x00000000

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface.

DWord	Bit	Description
0	1:0	**Priority** Format: U2 Value / Name: 00b — Highest priority 01b — Second highest priority 10b — Third highest priority 11b — Lowest priority

HW Generated BINDING_TABLE_STATE

HW Generated BINDING_TABLE_STATE		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:0	**Surface State Pointer** Format: SurfaceStateOffset[21:6] [CHV, BSW]

Hword 1 Block Data Payload

MDP_HW1 - Hword 1 Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Hword** Project: All Format: U256 Specifies the Hword data

Hword 2 Block Data Payload

MDP_HW2 - Hword 2 Block Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Hword0** Project: All Format: U256 Specifies the Hword data for element 0
1.0-1.7	255:0	**Hword1** Project: All Format: U256 Specifies the Hword data for element 1

Hword 4 Block Data Payload

MDP_HW4 - Hword 4 Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	1024	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Hword0** Project: All Format: U256 Specifies the Hword data for element 0
1.0-1.7	255:0	**Hword1** Project: All Format: U256 Specifies the Hword data for element 1
2.0-2.7	255:0	**Hword2** Project: All Format: U256 Specifies the Hword data for element 2
3.0-3.7	255:0	**Hword3** Project: All Format: U256 Specifies the Hword data for element 3

Hword 8 Block Data Payload

MDP_HW8 - Hword 8 Block Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Hword0** Project: All Format: U256 Specifies the Hword data for element 0
1.0-1.7	255:0	**Hword1** Project: All Format: U256 Specifies the Hword data for element 1
2.0-2.7	255:0	**Hword2** Project: All Format: U256 Specifies the Hword data for element 2
3.0-3.7	255:0	**Hword3** Project: All Format: U256 Specifies the Hword data for element 3
4.0-4.7	255:0	**Hword4** Project: All

MDP_HW8 - Hword 8 Block Data Payload		
		Format: U256 Specifies the Hword data for element 4
5.0-5.7	255:0	**Hword5** Project: All Format: U256 Specifies the Hword data for element 5
6.0-6.7	255:0	**Hword6** Project: All Format: U256 Specifies the Hword data for element 6
7.0-7.7	255:0	**Hword7** Project: All Format: U256 Specifies the Hword data for element 7

Hword Channel Mode Message Header Control

MHC_A64_CMODE - Hword Channel Mode Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Reserved** Project: All Format: MDC_CMODE [CHV, BSW] Specifies whether the read or write operation occurs on all 4 Dwords if any of those channel enables are set, or else only on the dwords whose corresponding channel enable is set.
	30:0	**Reserved** Project: All Format: Ignore Ignored

Hword Register Blocks Message Descriptor Control Field

MDC_DB_HW - Hword Register Blocks Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2
Default Value:	0x00000000

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>1:0</td><td>Register Blocks
<table>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Format:</td><td>Enumeration</td></tr>
</table>
Specifies the number of Hword blocks to be read or written
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00h</td><td>HW1</td><td>1 Hword register</td><td>All</td></tr>
<tr><td>01h</td><td>HW2</td><td>2 Hword registers</td><td>All</td></tr>
<tr><td>02h</td><td>HW4</td><td>4 Hword registers</td><td>All</td></tr>
<tr><td>03h</td><td>HW8</td><td>8 Hword registers</td><td>All</td></tr>
</table>
</td></tr>
</table>

Ignored Message Header

MH_IGNORE - Ignored Message Header	
Project:	CHV, BSW
Source:	DataPort 0
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Some messages require a message header or have an optional message header, but do not use any information in the header.

DWord	Bit	Description
0-7	255:0	**Reserved** Project: All Format: Ignore Ignored

Inline Data Description for MFD_AVC_BSD_Object

Inline Data Description for MFD_AVC_BSD_Object

Project: CHV, BSW
Source: VideoCS
Size (in bits): 96
Default Value: 0x00000000, 0x00000000, 0x00000000

This structure includes all the required Slice Header parameters and error handling settings for AVC_BSD_OBJECT Command (DW3..DW5).

DWord	Bit	Description
0	31	**Concealment Method** This field specifies the method used for concealment when error is detected. If set, a copy from collocated macroblock location is performed from the concealment reference indicated by the ConCeal_Pic_Id field. If it is not set, a copy from the current picture is performed using Intra 16x16 Prediction method. Value / Name / Description: 0 — Intra 16x16 Prediction 1 — Inter P Copy
	30	**Init Current MB Number** When set, the current Slice_Start_MB_Num, Slice_MB_Start_Hor_Pos and Slice_MB_Start_Vert_Pos fields will be used to initialize the Current_MB_Number register. This effectively disables the concealment capability.
	29	**Intra PredMode (4x4/8x8 Luma) Error Control Bit** Project: CHV, BSW This field controls if AVC decoder will fix Intra Prediction Mode if the decoded value is incorrect according to MB position Value / Name / Description: 0 — AVC decoder will detect and fix IntraPredMode (4x4/8x8 Luma) Errors. 1 — AVC decoder will NOT detect IntraPredMode (4x4/8x8 Luma) Errors. The wrong IntraPredMode value will be retaind.
	28:27	**MB Error Concealment B Temporal Prediction mode** These two bits control how the reference L0/L1 are overridden in B temporal slice. Value / Name / Description: 00b — **[Default]** — Both Reference Indexes L0/L1 are forced to 0 during Concealment 01b — Only Reference Index L1 is forced to 0; Reference Index L0 is forced to -1 10b — Only Reference Index L0 is forced to 0; Reference Index L1 is forced to -1 11b — Reserved — Invalid
	26	**Reserved** Project: CHV, BSW

Inline Data Description for MFD_AVC_BSD_Object

	Format:	MBZ

25 MB Error Concealment B Temporal Motion Vectors Override Enable Flag

During MB Error Concealment on B slice with Temporal Direct Prediction, motion vectors are forced to 0 to improve image quality. This bit can be set to preserve the original weight prediction.

Value	Name	Description
0	**[Default]**	Predicted Motion Vectors are used during MB Concealment
1		Motion Vectors are Overridden to 0 during MB Concealment

24 MB Error Concealment B Temporal Weight Prediction Disable Flag

During MB Error Concealment on B slice with Temporal Direct Prediction, weight prediction is disabled to improve image quality. This bit can be set to preserve the original weight prediction.

Value	Name	Description
0	**[Default]**	Weight Prediction is Disabled during MB Concealment
1		Weight Prediction will not be overridden during MB Concealment

23:22 Reserved

Format:	MBZ

21:16 Concealment Picture ID

This field identifies the picture in the reference list to be used for concealment. This field is only valid if **Concealment Method** is Inter P Copy.

Bit Filed	Value	Defenition
21	0	Frame Picture
21	1	Field picture
20:16	All	Frame Store Index[4:0]

15 Reserved

Format:	MBZ

14 BSD Premature Complete Error Handling

BSD Premature Complete Error occurs in situation where the Slice decode is completed but there are still data in the bitstream.

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W)
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically performs the error handling

13 Reserved

Format:	MBZ

12 MPR Error (MV out of range) Handling

Software must follow the action for each Value as follow:

Value	Name	Description

Inline Data Description for MFD_AVC_BSD_Object

	1		Set the interrupt to the driver (provide MMIO registers for MB address R/W)
	0		Ignore the error and continue (masked the interrupt), assume the hardware automatically performs the error handling

11 — Reserved

Format:	MBZ

10 — Entropy Error Handling

Software must follow the action for each Value as follow:

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W).
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error handling.

9 — Reserved

Format:	MBZ

8 — MB Header Error Handling

Software must follow the action for each Value as follow:

Value	Name	Description
1		Set the interrupt to the driver (provide MMIO registers for MB address R/W).
0		Ignore the error and continue (masked the interrupt), assume the hardware automatically perform the error concealment.

7:6 — MB Error Concealment B Spatial Prediction mode

These two bits control how the reference L0/L1 are overridden in B spatial slice.

Value	Name	Description
00b	**[Default]**	Both Reference Indexes L0/L1 are forced to 0 during Concealment
01b		Only Reference Index L1 is forced to 0; Reference Index L0 is forced to -1
10b		Only Reference Index L0 is forced to 0; Reference Index L1 is forced to -1
11b	Reserved	Invalid

5 — Reserved

Project:	CHV, BSW
Format:	MBZ

4 — MB Error Concealment B Spatial Motion Vectors Override Disable Flag

During MB Error Concealment on B slice with Spatial Direct Prediction, motion vectors are forced to 0 to improve image quality. This bit can be set to use the predicted motion vectors instead. This bit does not affect normal decoded MB.

Value	Name	Description
0	**[Default]**	Motion Vectors are Overridden to 0 during MB Concealment
1		Predicted Motion Vectors are used during MB Concealment

3 — MB Error Concealment B Spatial Weight Prediction Disable Flag

During MB Error Concealment on B slice with Spatial Direct Prediction, weight prediction is

Inline Data Description for MFD_AVC_BSD_Object

disabled to improve image quality. This bit can be set to preserve the original weight prediction. This bit does not affect normal decoded MB.

Value	Name	Description
0	**[Default]**	Weight Prediction is Disabled during MB Concealment.
1		Weight Prediction will not be overridden during MB Concealment.

2 — Reserved

Project:	CHV, BSW
Format:	MBZ

1 — MB Error Concealment P Slice Motion Vectors Override Disable Flag

During MB Error Concealment on P slice, motion vectors are forced to 0 to improve image quality. This bit can be set to use the predicted motion vectors instead. This bit does not affect normal decoded MB.

Value	Name	Description
0	**[Default]**	Motion Vectors are Overridden to 0 during MB Concealment
1		Predicted Motion Vectors are used during MB Concealment

0 — MB Error Concealment P Slice Weight Prediction Disable Flag

During MB Error Concealment on P slice, weight prediction is disabled to improve image quality. This bit can be set to preserve the original weight prediction. This bit does not affect normal decoded MB.

Value	Name	Description
0	**[Default]**	Weight Prediction is Disabled during MB Concealment.
1		Weight Prediction will not be overridden during MB Concealment.

1

31:16 — First MB Byte Offset of Slice Data or Slice Header

Programming Notes	Project
MFX supports only DXVA2 Long and Short Format.	CHV, BSW

15:8 — Reserved

Format:	MBZ

7 — Fix Prev Mb Skipped

Enables an alternative method for decoding mb_skipped, to cope with an encoder that codes a skipped MB as a direct MB with no coefficient.

6:5 — Reserved

Format:	MBZ

Programming Notes
Please note that the field MUST be set to '0' at this time.

4 — Emulation Prevention Byte Present

Value	Name	Description
0		H/W needs to perform Emulation Byte Removal

Inline Data Description for MFD_AVC_BSD_Object

		1		H/W does not need to perform Emulation Byte Removal
	3	**LastSlice Flag** It is needed for both error concealment at the end of a picture (so, no more phantom). It is also needed to know to set the last MB in a picture correctly. **Value** / **Name** / **Description** 1 /	If the current Slice to be decoded is the very last slice of the current picture. 0 /	If the current Slice to be decoded is any slice other than the very last slice of the current picture
	2:0	**First Macroblock (MB)Bit Offset** Exists If: //AVC Long Format Only Format: U3 This field provides the bit offset of the first macroblock of the Slice in the first byte of the input compressed bitstream.
2 **Project:** CHV, BSW	31	**I Slice Concealment Mode** Project: CHV, BSW This field controls how AVC decoder handle MB concealment in I Slice **Value** / **Name** 0 / Intra Concealment 1 / Inter Concealment **Programming Notes** If this field is set to "1" (Inter Concealment), driver must provide a valid reference picture (programmed using "Concealment Reference Picture" field) for concealment reference picture. In this mode, weight prediction is disabled and motion vectors are forced to 0 as well.
	30	**Reserved** Project: CHV, BSW Format: MBZ
	29:24	**Concealment Reference Picture + Field Bit** Project: CHV, BSW Format: U6 This field provides the concealment reference picture for hardware to conceal in case driver wants to specify one concealment picture. This field matches with the DPB order sent to hardware. This field applies to all I/P/B slices **Bit Filed** / **Value** / **Defenition** 29 / MBZ / is reserved for future expansion 28:25 / All / Reference PIcture Number

Inline Data Description for MFD_AVC_BSD_Object

24	All	Field Bit(if the current picture is a field picture [Frame picture must be 0]

23 P Slice Concealment Mode

Project:	CHV, BSW

This field controls how AVC decoder handle MB concealment in P Slice

Value	Name
1	Intra Concealment
0	Inter Concealment

22:19 Reserved

Project:	CHV, BSW
Format:	MBZ

18:16 P Slice Inter Concealment Mode

Project:	CHV, BSW

This field controls how AVC decoder select reference picture for Concealment in P Slice.

Value	Name	Description
000b		Top of Reference List L0 (Use top entry of Reference List L0)
001b		Driver Specified Concealment Reference
010b		Predicted Reference (Use reference picture predicted using P-Skip Algorithm)
011b		Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC]
100b		First Long Term Picture in Reference List L0 (If no long term picture available, use Temporal Closest Picture)
101b-111b	Reserved	

15 B Slice Concealment Mode

Project:	CHV, BSW

This field controls how AVC decoder handle MB concealment in B Slice

Value	Name
1	Intra Concealment
0	Inter Concealment

14 Reserved

Project:	CHV, BSW
Format:	MBZ

13:12 B Slice Inter Direct Type Concealment Mode

Project:	CHV, BSW

AVC decoder can use Spatial or Temporal Direct for B Skip/Direct. This field determine can override the mode on how AVC decoder handles MB concealment in B slice.

Inline Data Description for MFD_AVC_BSD_Object

Value	Name	Description
00b		Use Default Direct Type (slice programmed direct type)
01b		Forced to Spatial Direct Only
10b		Forced to Temporal Direct Only
11b		Spatial Direct without Temporal Componenet (MovingBlock information)

11 **Reserved**

Project:	CHV, BSW
Format:	MBZ

10:8 **B Slice Spatial Inter Concealment Mode**

Project:	CHV, BSW

This field controls how AVC decoder select reference picture for Spatial Inter Concealment in B Slice.

Value	Name	Description
000b		Top of Reference List L0/L1 (Use top entry of Reference List L0/L1).
001b		Driver Specified Concealment Reference
011b		Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC] [For L1: Closest POC larger than current POC]
100b		" First Long Term Picture in Reference List L0/L1 (If no long term picture available, use Temporal Closest Picture)
101b-111b	Reserved	

7 **Reserved**

Project:	CHV, BSW
Format:	MBZ

6:4 **B Slice Temporal Inter Concealment Mode**

Project:	CHV, BSW

This field controls how AVC decoder select reference picture for Temporal Inter Concealment in B Slice

Value	Name	Description
000b		Top of Reference List L0/L1 (Use top entry of Reference List L0/L1)
001b		Driver Specified Concealment Reference
010b		Predicted Reference (Use reference picture predicted using B-Skip Algorithm)
011b		" Temporal Closest (Using POC to select the closest forward picture) [For L0: Closest POC smaller than current POC] [For L1: Closest POC larger than current POC]
100b		First Long Term Picture in Reference List L0/L1 (If no long term picture

Inline Data Description for MFD_AVC_BSD_Object

		available, use Temporal Closest Picture)
101b-111b	Reserved	

3:2 Reserved

Project:	CHV, BSW
Format:	MBZ

1 Intra 8x8/4x4 Prediction Error Concealment Control Bit

Project:	CHV, BSW

This field controls if AVC goes into MB concealment mode (next MB) when an error is detected on Intra8x8/4x4 Prediction Mode (these 2 modes have fixed coding so it may not affect the bitstream.

Value	Name	Description
0		AVC decoder will NOT go into MB concealment when Intra8x8/4x4 Prediction mode is incorrect.
1		AVC decoder will go into MB concealment when Intra8x8/4x4 Prediction mode is incorrect.

0 Intra Prediction Error Control Bit (applied to Intra16x16/Intra8x8/Intra4x4 Luma and Chroma)

Project:	CHV, BSW

This field controls if AVC decoder will fix Intra Prediction Mode if the decoded value is incorrect according to MB position.

Value	Name	Description
0		AVC decoder will detect and fix Intra Prediction Mode Errors.
1		AVC decoder will retain the Intra Prediction value decoded from bitstream.

Inline Data Description - VP8 PAK OBJECT

Inline Data Description - VP8 PAK OBJECT

Project:	CHV, BSW
Source:	VideoCS
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure corresponds to Dw3..6 of MFX_VP8_PAK_OBJECT Command.

DWord	Bit	Description
0	31:23	**Reserved** Format: MBZ
	22:20	**MV Format(Motion Vector Size)** Exists If: //IntraMbFlag = 0 This field specifies the size and format of the output motion vectors. Value / Name / Description: 000b – Intra MB – No Motion vectors 100b – Inter Predict MB (Unpacked Motion Vector Mode) – Sixteen Motion Vectors Per MacroBlock Others – Reserved **Programming Notes** This field MBZ, when the **IntraMbFlag = 1**.
	19:18	**SegmentID** Format: U2 Segment number 0-3
	17	**Enable Coeff Clamp** Value / Name / Description: 1 – Magnitude of coefficients of the current MB is clamped based on the clamping matrix after quantization 0 – No Clamping
	16:14	**Reserved** Format: MBZ
	13	**Intra MB Flag** This field specifies whether the current macroblock is an Intra (I) Macroblock. For Key pictures (IsKyeFrameFlag DW2, bit[5] of MFX_VP8_PIC_STATE), this field must be set to 1. Value / Name: 0h – INTER (Inter MacroBlock)

Inline Data Description - VP8 PAK OBJECT

		1h: INTRA (Intra MacroBlock)

Programming Notes
For I-picture MB (Intra MB Flag =1), this field must be set to 1.

12:11 **RefPicSelect**

This field specifies which reference pic (among Last Frame, Golden Frame and Alt Frame) is selected for the current macroblock when Intra MB Flag = 0 .

Value	Name
00b	Last Frame
01b	Golden Frame
10b	Alt Frame

10:8 **MB Type 3-Bits - Inter/Intra MB**

MB Type 3 Bits [10:8] specifies InterMB MV mode configurations: 16x16 or 2 16x8 or 4 8x8 or 16 4x4 when Intra MB Flag = 0 and bit [8] = IntraMB mode configurations: 4x4 or 16x16 when Intra MB Flag = 1

Value	Name	Description
000b	16x16	**Inter MB** Only DW 6 bits 3:0 are used to indicate MVMode, MVMode can't be split
001b	2 16x8 (mv_Top Bottom)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 16x8 partition, DW6 bits[3:0] are used for MVMode for second 16x8 partition.
010b	2 8 x16 (mv_left_right)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 8x16 partition, DW5 bits[11:8] are used for MVMode for second 8x16 partition.
011b	4 8x8 (mv_quarters)	**Inter MB [10:8]** Split MV is inferred. DW5 bits[3:0] are used for MVMode for first 8x8 partition. DW5 bits[11:8] are used for MvMode for second 8x8 partition. DW6 bits[3:0] are used for MVMode for third 8x8 partition. DW6 bits[11:8] are used for MVMode for fourth 8x8 partition.
100b	16 4x4 (mv_16)	**Inter MB [10:8]** Split MV is inferred. There are 16 partitions. Each Sub-block uses 4 bits in DW6 and DW7.
0b	16x16	**Intra MB [8]** Only DW5, bits[3:0] are used for Y mode. For B_PRED, "16 4x4" should be used which implies B_PRED mode.
1b	16 4x4	**Intra MB [8]** All bits in DW5 and DW6 are used to represent B_PRED modes (Bmodes) in each sub-blocks.

7:6 **Reserved**

Format:	MBZ

5:4 **MB UV Mode**

Value	Name
0	DC_PRED

Inline Data Description - VP8 PAK OBJECT

		1 V_PRED 2 H_PRED 3 TM_PRED
	3	**Reserved** Format: MBZ
	2	**Skip MB Flag** This field is equivalent to mb_skip_flag in VP8 spec. **Programming Notes** By setting this field to 1, it forces an Inter MacroBlock to be encoded as a skipped MacroBlock
	1:0	**Reserved** Format: MBZ
1	31:24	**Reserved** Format: MBZ
	23:16	**MbYCnt (Vertical Origin)** Format: U8 Unit of MacroBlock This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks.
	15:8	**Reserved** Format: MBZ
	7:0	**MbXCnt (Horizontal Origin)** Format: U8 Unit of MacroBlock This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks.
2	31:28	**B Mode for SubBlock7 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	27:24	**B Mode for SubBlock6 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	23:20	**B Mode for SubBlock5 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	19:16	**B Mode for SubBlock4 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	15:12	**B Mode for SubBlock3 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	11:8	**B Mode for SubBlock2 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	7:4	**B Mode for SubBlock1 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.

Inline Data Description - VP8 PAK OBJECT		
	3:0	**B Mode for SubBlock0 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
3	31:28	**B Mode for SubBlock15 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	27:24	**B Mode for SubBlock14(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	23:20	**B Mode for SubBlock13(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	19:16	**B Mode for SubBlock12(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	15:12	**B Mode for SubBlock11(Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	11:8	**B Mode for SubBlock10 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	7:4	**B Mode for SubBlock9 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.
	3:0	**B Mode for SubBlock8 (Y mode for the macroblock in non-B mode)** For Y-Mode and B-Mode Assignments refer to the assignment lists below this table.

INTERFACE_DESCRIPTOR_DATA

INTERFACE_DESCRIPTOR_DATA

Project: CHV, BSW

Source: RenderCS

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:6	**Kernel Start Pointer** Format: InstructionBaseOffset[31:6]Kernel Specifies the 64-byte aligned address offset of the first instruction in the kernel. This pointer is relative to the **Instruction Base Address.**
	5:0	**Reserved** Format: MBZ
1	31:16	**Reserved** Format: MBZ
	15:0	**Kernel Start Pointer High** Format: InstructionBaseOffset[47:32]Kernel This field specifies the high 16 bits of starting address of the Kernel Pointer.
2	31:20	**Reserved** Project: CHV, BSW Format: MBZ
	19	**Denorm Mode** This field specifies how Float denormalized numbers are handles in the dispatched thread. Value / Name / Description: 0h / Ftz / Float denorms will be flushed to zero when appearing as inputs, denorms will never come out of instructions. Double precision float and half precision float numbers are not flushed to zero. 1h / SetByKernel / Denorms will be handled in by kernel.
	18	**Single Program Flow** Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). Value / Name: 0h / Multiple 1h / Single
	17	**Thread Priority**

INTERFACE_DESCRIPTOR_DATA

Specifies the priority of the thread for dispatch.

Value	Name
0h	Normal Priority
1h	High Priority

16 Floating Point Mode

Specifies the floating point mode used by the dispatched thread.

Value	Name
0h	IEEE-754
1h	Alternate

15:14 Reserved

Format:	MBZ

13 Illegal Opcode Exception Enable

Format:	Enable

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

12 Reserved

Format:	MBZ

11 Mask Stack Exception Enable

Format:	Enable

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

10:8 Reserved

Format:	MBZ

7 Software Exception Enable

Format:	Enable

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

6:0 Reserved

Format:	MBZ

3 31:5 Sampler State Pointer

Format:	DynamicStateOffset[31:5]SAMPLER_STATE

Specifies the 32-byte aligned address offset of the sampler state table. This pointer is relative to the **Dynamic State Base Address.***This field is ignored for child threads.*

4:2 Sampler Count

Format:	U3

INTERFACE_DESCRIPTOR_DATA

Specifies how many samplers (in multiples of 4) the kernel uses. Used only for prefetching the associated sampler state entries. *This field is ignored for child threads.If this field is not zero, sampler state is prefetched for the first instance of a root thread upon the startup of the media pipeline.*

Value	Name
[0,4]	
0h	No samplers used
1h	Between 1 and 4 samplers used
2h	Between 5 and 8 samplers used
3h	Between 9 and 12 samplers used
4h	Between 13 and 16 samplers used

1:0 Reserved

Format:	MBZ

4 — 31:16 Reserved

Format:	MBZ

15:5 Binding Table Pointer

Format:	SurfaceStateOffset[15:5]BINDING_TABLE_STATE*256

Specifies the 32-byte aligned address of the binding table. This pointer is relative to the **Surface State Base Address**. *This field is ignored for child threads.*

4:0 Binding Table Entry Count

Project:	CHV, BSW
Format:	U5

Specifies how many binding table entries the kernel uses. Used only for prefetching of the binding table entries and associated surface state. *This field is ignored for child threads.If this field is not zero, binding table and surface state are prefetched for the first instance of a root thread upon the startup of the media pipeline.*

Value	Name
[0,31]	

Programming Notes
The maximum number of prefetched binding table entries is limited to 31. For kernels using a large number of binding table entries, it may be wise to set this field to zero to avoid prefetching too many entries and thrashing the state cache.

5 — 31:16 Constant/Indirect URB Entry Read Length

Format:	U16

Specifies the amount of URB data read and passed in the thread payload for the Constant or Indirect URB entry, in 8-DW register increments. A value 0 means that no Constant or Indirect

INTERFACE_DESCRIPTOR_DATA

URB Entry will be loaded. The Constant URB Entry Read Offset field will then be ignored. In GPGPU mode this describes how much data is delivered in a single dispatch. Multiple dispatches in a thread group will deliver constant data offset by this value. The total amount of constant data is (Constant URB Read Length * Number of Threads in GPGPU Thread Group + Cross-Thread Constant Data Read Length).

If **Cross-Thread Constant Data Read Length** for Indirect is greater than 0, then this field must also be greater than 0. The allowed combinations are:

Constant/Indirect URB Entry Read Length **Entry Read Length**	Cross-Thread Constant **Data Read Length**	Notes
=0	=0	No Payload
>0	=0	Per-thread payload only
>0	>0	Both kinds of payload
=0	>0	Only for CURBE payloads

Value	Name
[0,63]	

15:0 Constant URB Entry Read Offset

Format:	U16

Specifies the offset (in 8-DW units) at which Constant URB data is to be read from the URB before being included in the thread payload.

Value	Name	Description
[0,1983]		Indicating [0,1983] 256-bit register increments. ROB has 64KB of storage; 2048 entries. However, lowest 64 entries are reserved for VFE/TS to store interface descriptor data. Hence, (URB Entry Read Offset + Read Length) shall not exceed 1984.

6 31:24 Reserved

Format:	MBZ

23:22 Rounding Mode

Format:	U2

Value	Name	Description
00b	RTNE **[Default]**	Round to Nearest Even
01b	RU	Round toward +Infinity
10b	RD	Round toward -Infinity
11b	RTZ	Round toward Zero

21 Barrier Enable

INTERFACE_DESCRIPTOR_DATA

		Format: Enable This field specifies whether the thread group requires a barrier. If not, it can be dispatched without allocating one.
	20:16	**Shared Local Memory Size** Project: CHV, BSW Format: U5 This field indicates how much shared local memory the thread group requires. The amount is specified in 4k blocks, but only powers of 2 are allowed: 0, 4k, 8k, 16k, 32k and 64k per half-slice.

Value	Name
0	Encodes 0k
1	Encodes 4k
2	Encodes 8k
4	Encodes 16k
8	Encodes 32k
16	Encodes 64k

Programming Notes	Project
If SLMSize > 0, then a barrier must also be allocated.	CHV, BSW

	15	**Reserved** Project: CHV, BSW Format: MBZ
	14:10	**Reserved** Format: MBZ
	9:0	**Number of Threads in GPGPU Thread Group** Project: CHV, BSW Format: U10 Specifies the number of threads that are in this thread group. The minimum value is 1, while the maximum value is the number of threads in a subslice for local barriers. See vol1b Configurations for the number of threads per subslice for different products. The maximum value for global barriers is limited by the number of threads in the system, or by 511, whichever is lower. This field should not be set to 0 even if the barrier is disabled, since an accurate value is needed for proper pre-emption.
7	31:8	**Reserved** Format: MBZ
	7:0	**Cross-Thread Constant Data Read Length** Format: U8 Specifies the amount of constant data in CURBE in 8-DW register increments which will be sent

INTERFACE_DESCRIPTOR_DATA

to every thread in the thread group in addition to the per thread ids specified by **Constant URB Entry Read Length**.

Value	Name
[0,127]	

Invalidate After Read Message Descriptor Control Field

MDC_IAR - Invalidate After Read Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**Reserved** Project: All Format: MBZ Previously, this Enable field was intended to optimize scratch and spill/fill read messages, where the memory was only used by a single thread and did not need to be maintained after the thread completed. If enabled, it caused all lines in the L3 cache accessed by the message to be invalidated after the read occurred, regardless of whether the line contained modified data. It was intended as a performance hint indicating that the data would no longer be used to avoid writing back data to memory.

JPEG

JPEG		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:5	**Reserved** Format: MBZ
	4	**Inconsistent VLD SE Error** This flag indicates an inconsistent SE coded in the bit-stream. Bit-stream does not match any entries in the hauffman table.
	3	**Extra Block Error** This flag indicates extra block coded within an ECS data boundary.
	2	**Missing block Error** This flag indicates one or more blocks are missing within an ECS data boundary.
	1	**Extra ECS Error** This flag indicates extra ECS' coded in the bit-stream SCAN payload data.
	0	**Missing ECS Error** This flag indicates one or more ECS' are missing from the bit-stream SCAN payload data.

LOD Message Address Payload Control

MACD_LOD - LOD Message Address Payload Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:4	**Reserved** Project: All Format: MBZ Ignored
	3:0	**LOD** Project: All Format: U4 Specifies the LOD for this slot. Value | Name | Description [0,14] | | representing LOD

Lower Oword Block Data Payload

MDP_OW1L - Lower Oword Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword** Project: All Format: U128 Specifies the upper Oword data element
0.4-0.7	127:0	**Reserved** Project: All Format: Ignore Ignored

MEDIA_SURFACE_STATE

MEDIA_SURFACE_STATE

Project:	CHV, BSW
Source:	PRM
Exists If:	//([MessageType] == 'Deinterlace') OR ([MessageType] == 'Sample_8x8')
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the SURFACE_STATE used by only deinterlace, sample_8x8, and VME messages.

DWord	Bit	Description
0	31:0	**Reserved** Project: CHV, BSW Format: MBZ
1	31:18	**Height** Format: U14-1 This field specifies the height of the surface in units of pixels. For PLANAR surface formats, this field indicates the height of the Y (luma) plane. Value: [0,16383] — Name: — Description: representing heights [1,16384] — Exists If: [Surface Type] != FM_STRBUF_* **Programming Notes** Height (field value + 1) must be a multiple of 2 for PLANAR_420 surfaces.If Vertical Line Stride is 1, this field indicates the height of the field, not the height of the frame.
	17:4	**Width** Format: U14-1 This field specifies the width of the surface in units of pixels. For PLANAR surface formats, this field indicates the width of the Y (luma) plane. Value: [0,16383] — Name: — Description: representing widths [1,16384] — Exists If: [Surface Type] != FM_STRBUF_* **Programming Notes** • The Width specified by this field multiplied by the pixel size in bytes must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). • Width (field value + 1) must be a multiple of 2 for PLANAR_420, PLANAR_422, and all YCRCB_* and Y16_UNORM surfaces, and must be a multiple of 4 for PLANAR_411 and Y8_UNORM_VA surfaces. • For deinterlace messages, the Width (field value + 1) must be a multiple of 8.

MEDIA_SURFACE_STATE

- For Y8_UNORM_VA format width should be in multiple of 4, for Y16_UNORM_VA format width should be in multiple of 2, for Y1_UNORM format width should be in multiple of 32
- When Address Control = Mirror, the total width should be in multiple of 4bytes.

Width (field value + 1) must be a multiple of 2 for PLANAR_420_16

3:2 **Picture Structure**

Specifies the encoding of the current picture.

Value	Name
00b	Frame Picture
01b	Top Field Picture
10b	Bottom Field Picture
11b	Invalid, not allowed

1:0 **Cr(V)/Cb(U) Pixel Offset V Direction**

Default Value:	0
Format:	U0.2

Description	Project
Specifies the distance to the U/V values with respect to the even numbered Y channels in the V direction	CHV, BSW

Programming Notes	Project
This field is ignored for all formats except PLANAR_420_8	CHV, BSW

2 31:27 **Surface Format**

Project:	CHV, BSW

Specifies the format of the surface. All of the Y and G channels will use table 0 and all of the Cr/Cb/R/B channels will use table 1.

Note: Y8_UNORM_VA, Y16_UNORM and Y16_SNORM are used for all functions of sample_8x8 except AVS where rest of the formats are not used. These two formats are packed as 32bits in L1 though the individual pixels are either 8bpp or 16bpp respectively.

Value	Name	Description
4	PLANAR_420_8	
5	Y8_UNORM_VA	Sample_8x8 only except AVS
6	Y16_SNORM	Sample_8x8 only except AVS
7	Y16_UNORM_VA	Sample_8x8 only except AVS
8	R10G10B10A2_UNORM	Sample_8x8 only
9	R8G8B8A8_UNORM	Sample_8x8 AVS only

MEDIA_SURFACE_STATE

11	R8_UNORM (Cr/Cb)	Sample_8x8 AVS only
12	Y8_UNORM	Sample_8x8 AVS only
13	A8Y8U8V8_UNORM	Sample_8x8 AVS only
14	B8G8R8A8_UNORM	Sample_8x8 AVS only
15	R16G16B16A16	Sample_8x8 AVS only
16	Y1_UNORM	Sample_8x8 only for boolean surfaces (1bit/pixel)
Others	Reserved	

26 Interleave Chroma

Project:	CHV, BSW
Format:	Enable

Description
This field indicates that the chroma fields are interleaved in a single plane rather than stored as two separate planes. This field is only used for PLANAR surface formats.

25:22 Reserved

Project:	CHV, BSW
Format:	MBZ

21 Address Control

Project:	CHV, BSW

Value	Name	Description
0	CLAMP	Clamp
1	MIRROR	Mirror

20:3 Surface Pitch

Format:	U18-1 pitch in Bytes

This field specifies the surface pitch in (#Bytes - 1).

Value	Name	Description
[0,262143]		For other linear surfaces: representing [1B, 256KB]
[511, 262143]		For X-tiled surface: representing [512B, 256KB] = [1 tile, 512 tiles]
[127, 262143]		For Y-tiled surfaces: representing [128B, 256KB] = [1 tile, 2048 tiles]

Programming Notes
For tiled surfaces, the pitch must be a multiple of the tile widthIf Half Pitch for Chroma is set, this field must be a multiple of two tile widths for tiled surfaces, or a multiple of 2 bytes for linear surfaces.The Surface Pitches of current picture and reference picture should be declared as the identical type in VDI mode with identical Height, Width and Format.

2 Half Pitch for Chroma

Format:	Enable

MEDIA_SURFACE_STATE

This field indicates that the chroma plane(s) will use a pitch equal to half the value specified in the Surface Pitch field. This field is only used for PLANAR surface formats.

Programming Notes
Must be Zero as this field is not used.

1:0 **Tile Mode**

Format:	U2 Enumerated Type

This field specifies the type of memory tiling (Linear, WMajor, XMajor, or YMajor) employed to tile this surface. See Memory Interface Functions for details on memory tiling and restrictions.

Value	Name	Description	Project
0h	TILEMODE_LINEAR	Linear mode (no tiling)	All
1h	Reserved	Reserved	All
2h	TILEMODE_XMAJOR	X major tiling	All
3h	TILEMODE_YMAJOR	Y major tiling	All

Programming Notes

- Refer to *Memory Data Formats* for restrictions on TileMode direction for the various buffer types. (Of particular interest is the fact that YMAJOR tiling is not supported for display/overlay buffers).
- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this field.
- Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled (X/Y/W) surfaces can only be mapped to Main Memory.

3 31:30 **Reserved**

Project:	All
Format:	MBZ

29:16 **X Offset for U(Cb)**

Format:	U14 Pixel Offset

Description	Project
For non planar surfaces this field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the surface.	
For Planar surfaces this field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled.	CHV, BSW

Programming Notes
For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.

15:14 **Reserved**

MEDIA_SURFACE_STATE

		Format: MBZ
	13:0	**Y Offset for U(Cb)** Format: U14 Row Offset **Description** / **Project** For non planar surfaces this field specifies the vertical offset in pixels from the Surface Base Address to the start (origin) of the surface. / For Planar surfaces this field specifies the vertical offset in rows from the Surface Base Address to the start (origin) of the U(Cb) plane or the interleaved UV plane if Interleave Chroma is enabled. / CHV, BSW
4	31:30	**Reserved** Project: All Format: MBZ
	29:16	**X Offset for V(Cr)** Exists If: //([Surface Format] is one of planar) AND ([Interleave Chroma] == '0') Format: U14 Pixel Offset **Description** / **Project** This field specifies the horizontal offset in pixels from the Surface Base Address to the start (origin) of the V(Cr) plane. / CHV, BSW **Programming Notes** For PLANAR_420 and PLANAR_422 surface formats, this field must indicate an even number of pixels.
	15	**Reserved** Format: MBZ
	14:0	**Y Offset for V(Cr)** Exists If: //([Surface Format] is one of planar) AND ([Interleave Chroma] == '0') Format: U15 Row Offset **Description** / **Project** This field specifies the veritical offset in rows from the Surface Base Address to the start (origin) of the V(Cr) plane. / CHV, BSW **Programming Notes** / **Project** This field must indicate a multiple of 4 (bit 0 & 1 = 00). / CHV, BSW
5	31	**Vertical Line Stride** Project: CHV, BSW

MEDIA_SURFACE_STATE

		Format: U1 in lines to skip between logically adjacent lines For Surfaces accessed via the sample_8x8 message:Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.For Other Surfaces:Vertical Line Stride must be zero. **Workaround** / **Project** All surfaces used by the sampler between sampler cache invalidates must have the same setting of this field in both RENDER_SURFACE_STATE and MEDIA_SURFACE_STATE. / CHV, BSW
	30	**Vertical Line Stride Offset** Project: CHV, BSW Format: U1 in lines of initial offset (when Vertical Line Stride == 1) For Surfaces accessed via the sample_8x8 message: Specifies the offset of the initial line from the beginning of the buffer. For Other Surfaces: Vertical Line Stride Offset must be zero. **Programming Notes** / **Project** This field must be set to 0 if Vertical Line Stride is 0. / CHV, BSW
	29:24	**Reserved** Format: MBZ
	23:20	**Reserved** Project: CHV, BSW Format: MBZ
	19:18	**Reserved** Project: CHV, BSW Format: MBZ
	17:7	**Reserved** Format: MBZ
	6:0	**Surface Memory Object Control State** Default Value: 0h DefaultVaueDesc Project: CHV, BSW Format: MEMORY_OBJECT_CONTROL_STATE This 7-bit field is used in various state commands and indirect state objects to define cacheability and other attributes related to memory objects.
6	31:0	**Surface Base Address** Project: CHV, BSW Format: GraphicsAddress[31:0] Specifies the low 32 bits of the byte-aligned base address of the surface. **Programming Notes** For SURFTYPE_BUFFER render targets, this field specifies the base address of first element of the

MEDIA_SURFACE_STATE

		surface. The surface is interpreted as a simple array of that single element type. The address must be naturally-aligned to the element size (e.g., a buffer containing R32G32B32A32_FLOAT elements must be 16-byte aligned).For SURFTYPE_BUFFER non-rendertarget surfaces, this field specifies the base address of the first element of the surface, computed in software by adding the surface base address to the byte offset of the element in the buffer.Mipmapped, cube and 3D sampling engine surfaces are stored in a 'monolithic' (fixed) format, and only require a single address for the base texture.Linear render target surface base addresses must be element-size aligned, for non-YUV surface formats, or a multiple of 2 element-sizes for YUV surface formats. Other linear surfaces have no alignment requirements (byte alignment is sufficient.)Linear depth buffer surface base addresses must be 64-byte aligned. Note that while render targets (color) can be SURFTYPE_BUFFER, depth buffers cannot.Tiled surface base addresses must be 4KB-aligned. Note that only the offsets from Surface Base Address are tiled, Surface Base Address itself is not transformed using the tiling algorithm.For tiled surfaces, the actual start of the surface can be offset from the Surface Base Address by the X Offset and Y Offset fields.Certain message types used to access surfaces have more stringent alignment requirements. Please refer to the specific message documentation for additional restrictions.
7	31:16	**Reserved** Format: MBZ
	15:0	**Surface Base Address High** Project: CHV, BSW Format: GraphicsAddress[47:32] Specifies the high 16 bits of the byte-aligned base address of the surface. Refer to Surface Base Address [31:0] for programming notes applying to this field.

MEMORY_OBJECT_CONTROL_STATE

MEMORY_OBJECT_CONTROL_STATE

Project: CHV, BSW
Source: PRM
Size (in bits): 7
Default Value: 0x00000000

DWord	Bit	Description
0	6:5	**Memory Type** Defines the memory type used in caching accesses to target caches. Value / Name: 00b – UC 01b – Reserved 10b – Reserved 11b – WB
	4:3	**Target Cache** Allows the control to target caching in the GFX pipeline for EU related surfaces. Value / Name: 00b – No Caching 01b – No Caching 1 10b – No Caching 2 11b – L3 Cache Allowed
	2	**Reserved**
	1:0	**Reserved** Format: MBZ

MemoryAddressAttributes

MemoryAddressAttributes

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

This field controls the priority of arbitration used in the GAC/GAM pipeline for this surface. It defines the CHV, BSW 32-bit memory address attributes for the third DWord of the HCP command buffer address.

DWord	Bit	Description
0 **Project:** CHV, BSW	31:9	**Reserved** Project: CHV, BSW Format: MBZ
	8:7	**Base Address - Arbitration Priority Control** Project: CHV, BSW Format: HEVC_ARBITRATION_PRIORITY [CHV, BSW]
	6:5	**Reserved** Project: CHV, BSW
	4:3	**Base Address - Target Cache (TC)** Project: CHV, BSW Format: U2 This field allows the choice of LLC vs. eLLC for caching. Value / Name / Description: 00b / eLLC Only / Not snooped in GT 01b / LLC Only / 10b / LLC/eLLC Allowed / 11b / L3, LLC, eLLC Allowed /
	2	**Reserved** Project: CHV, BSW
	1:0	**Base Address - Age for QUADLRU (AGE)** Project: CHV, BSW Format: U2 This field allows the selection of AGE parameter for a given surface in LLC. If a particular allocation is done at youngest age ("3") it tends to stay longer in the cache as compared to older age allocations ("2", "1", or "0"). This option is given to the driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches.

MemoryAddressAttributes

This field is also used for eLLC.

Value	Name
00b	Good chance of generating hits
01b	Next good chance of generating hits
10b	Decent chance of generating hits
11b	Poor chance of generating hits

Merged Media Block Message Header

MH_MBM - Merged Media Block Message Header

Project: CHV, BSW
Source: DataPort 1
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Project: All Format: S31 X offset (in bytes) of the upper left corner of the block into the surface.
1	31:0	**Y Offset** Project: All Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Merged Media Block Message Control** Project: All Format: MHC_MBM_CONTROL [CHV, BSW] Specifies the Merged message subtype and additional input parameters.
3	31:0	**Mask** Project: All Format: U32 The Mask is ignored by the Merged Media Block message: all Dwords are always returned on reads, and always enabled to be written on writes.
4	31:0	**FFTID** Project: All Format: MHC_FFTID [CHV, BSW] Fixed Function Thread ID
5-7	95:0	**Reserved** Project: All Format: Ignore Ignored

MH_MBM - Merged Media Block Message Header		

Merged Media Block Message Header Control

MHC_MBM_CONTROL - Merged Media Block Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Project: All Format: Enumeration Specifies the Media Block Read message is Normal subtype. Value / Name / Description / Project: 00h / Normal / The Block Height and Block Width fields are specified in this Dword. The Mask is ignofed by a media block read message. / All Others / Reserved / Reserved. / All
	29	**Reserved** Project: All Format: Ignore Ignored
	28:24	**Sub-Register Offset** Project: All Format: U5 Provides the sub-register offset in unit of bytes of a Merged Media Block Read message. This field is ignored (reserved) for a media block write message. Range = [0, 28]. Only a multiple of BasePitch, including 0, is valid. **Programming Notes** Sub-Register Offset and Register Pitch Control allow software to assembly multiple media block reads directly into a shared GRF register set. For example, if both are set to zero, the read data are written to GRF registers, aligning to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the Block Width. If Register Pitch Control is non-zero, multiple media block read messages sharing the same Register Pitch Control but with different Sub-Register Offset can fill in the same set of GRF registers with media block data line interleaved. **Restriction** For the Sampler Cache Data, this field must be zero.

MHC_MBM_CONTROL - Merged Media Block Message Header Control

BasePitch is defined as the next the power-of-2 that is greater than or equal to the Block Width. Minimum BasePitch is 1 DWord.
Sub-Register Offset must be aligned to BasePitch (therefore will be a multiple of DWords as well). When Register Pitch Control = 0, Sub-Register Offset must align to BasePitch*Block Height. ensuring the output fits in a single GRF register. In general (and specifically when Sub-Register Offset is greater than 0), when the resulting data will cross a GRF register boundary, the data must be placed symmetrically between GRF registers.

23:22 Reserved

Project:	All
Format:	Ignore

Ignored

21:16 Block Height

Project:	All
Format:	U6

Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows

Restriction
If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) * (# rows) <= 64 Dwords.

15:10 Reserved

Project:	All
Format:	Ignore

Ignored

9:8 Reserved

Project:	CHV, BSW*:A0
Format:	MBZ

Restriction : Must be zero.

7:6 Reserved

Project:	All
Format:	Ignore

Ignored

5:0 Block Width

Project:	All
Format:	U6

Width in bytes of the block being accessed. Range = [0,31] representing 1 to 32 Bytes.

MHC_MBM_CONTROL - Merged Media Block Message Header Control

Message Descriptor - Render Target Write

Message Descriptor - Render Target Write

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31	**Reserved** Format: MBZ
	30	**Data Format** Project: CHV, BSW Format: U1 Value / Name / Description: 0 – Single Precision – 32b 1 – Half Precision – 16b **Programming Notes** This field is applicable for Render Target Write Messages ONLY.
	29:14	**Reserved** Format: MBZ
	13	**Reserved** Project: CHV, BSW Format: MBZ
	12	**Last Render Target Select** This bit must be set on the last render target write message sent for each group of pixels. For single render target pixel shaders, this bit is set on all render target write messages. For multiple render target pixel shaders, this bit is set only on messages sent to the last render target. This bit must be zero for SIMD8 Image Write message. **Programming Notes** In general, when threads are not launched by 3D FF, this bit must be zero.
	11	**Slot Group Select** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the antialias alpha, multisample coverage mask, and if the header is not present also includes the X/Y addresses and pixel enables. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed. Value / Name / Description

Message Descriptor - Render Target Write

0	SLOTGRP_LO	choose bypassed data for slots 15:0
1	SLOTGRP_HI	choose bypassed data for slots 31:16

Programming Notes
For SIMD8 Image Write message thsi field MBZ.

10:8 **Message Type**

This field specifies the type of render target message. For the SIMD8_DUALSRC_xx messages, the low bit indicates which slots to use for the pixel enables, X/Y addresses, and oMask.

Value	Name	Description
000b	SIMD16	SIMD16 single source message
001b	SIMD16_REPDATA	SIMD16 single source message with replicated data
010b	SIMD8_DUALSRC_LO	SIMD8 dual source message, use slots 7:0
011b	SIMD8_DUALSRC_HI	SIMD8 dual source message, use slots 15:8
100b	SIMD8_LO	SIMD8 single source message, use slots 7:0
111b	SIMD16_REPDATA	It's only supported when accessing *Tiled Memory*. Using this Message Type to access linear *(Untiled)* memory is UNDEFINED.

Programming Notes	Project
the above slots indicated are within the 16 slots selected by **Slot Group Select**. If SLOTGRP_HI is selected, the SIMD8 message types above reference slots 23:16 or 31:24 instead of 7:0 or 15:8, respectively.	
SIMD16_REPDATA message must not be used in SIMD8 pixel-shaders.	CHV, BSW

7:0 **Reserved**

Format:	MBZ

Message Descriptor - Sampling Engine

Message Descriptor - Sampling Engine

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**EOT** Project: All
	30	**Return Format** Project: CHV, BSW Format: U1 Value / Name / Description 0 / 32-bit / Return data is 32b 1 / 16-bit / Return data is 16b **Programming Notes** This field must be set to 32-bit for messages with SIMD Mode of SIMD4x2 or SIMD32/64. This field must be set to 32 for resinfo, LOD and sampleinfo messages.
	29	**Reserved** Project: CHV, BSW Format: MBZ
	28:25	**Message Length** Format: U4 This field specifies the number of 256-bit GRF registers starting from (src) to be sent out on the request message payload. Value / Name [1,15] **Programming Notes** A value of 0 is considered erroneous.
	24:20	**Response Length** Format: U5 This field indicates the number of 256-bit registers expected in the message response. Value / Name [0,16]

Message Descriptor - Sampling Engine

		Programming Notes A value 0 indicates that the request message does not expect any response. The largest response supported is 16 GRF registers.
	19	**Header Present** Format: Enable Specifies whether the message includes a header phase. If the header is not present (this field is zero), all of the fields normally contained in the header are assumed to be 0.
	18:17	**SIMD Mode[1:0]** Format: U2 Specifies the SIMD mode of the message being sent.
	16:12	**Message Type** Format: U5 Specifies the type of message being sent. For more details, please refer to **Message Format** section for the definition of these 5 bits..
	11:8	**Sampler Index** Format: U4 Specifies the index into the sampler state table. Ignored for ld, resinfo, sampleinfo, and cache_flush type messages. **Value**: [0,15] **Programming Notes** • For the deinterlace message, this field must be a multiple of 2 (even). • For the sample_8x8 message, this field must be a multiple of 4.
	7:0	**Binding Table Index** Format: U8 Specifies the index into the **binding table**. Ignored for cache_flush type messages. Values of 255 and 253 indicate stateless. 254 indicates SLM. 252 indicates bindless. **Value**: [0,255]

MFD_MPEG2_BSD_OBJECT Inline Data Description

MFD_MPEG2_BSD_OBJECT Inline Data Description

Project:	CHV, BSW
Source:	VideoCS
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

DW0..1 corresponds to DW3..4 of the MFD_MPEG2_BSD_OBJECT.

DWord	Bit	Description
0	31:24	**Slice Horizontal Position** Format: U8 in Macroblocks This field indicates the horizontal position of the first macroblock in the slice.
	23:16	**Slice Vertical Position** Format: U8 in Macroblocks This field indicates the vertical position of the first macroblock in the slice.
	15:8	**Macroblock Count** Format: U8 in Macroblocks This field indicates the number of macroblocks in the slice, including skipped macroblocks.
	7	**Slice Concealment Override Bit** This bit forces hardware to handle the current slice in Conceal or Deocde Mode. If this bit is set to one, VIN will force the current slice to do concealment or to decode from bitstream regardless if the slice boundary has errors or not. Value / Name / Description: 1h — VIN will use driver-provided "Slice Concealment Type" regardless of valid slice boundary 0h — Driver must program "Slice Concealment Type" to '0'. VIN will set "Slice Concealment Type" depending if the slice boundary has error or not
	6	**Slice Concealment Type Bit** This bit can be forced by driver ("Slice Concealment Override Bit") or set by VINunit depending on slice boundary errors. Value / Name / Description: 1h — VMD will conceal all MBs of the slice regardless of bitstream. (If driver does not force the value of this bit, VIN will set this bit depending on slice boundary error. If the next slice position of the current slice is out-of-bound or the same or earlier than the current slice start position, VIN will set this bit for the next slice) 0h — VMD will decode MBs from the bitstream until the bitstream is run-out. Then VMD will conceal the remaining MBs.

MFD_MPEG2_BSD_OBJECT Inline Data Description

		Programming Notes VIN can turn this bit from 0 to 1 internally if "Slice Concealment Disable Bit" is "0" and VIN detects slice boundary errors.
	5	**Last Pic Slice** This bit is added to support error concealment at the end of a picture. Value / Name / Description: 1h — The current Slice is the last Slice of the entire picture 0h — The current Slice is not the last Slice of current picture
	4	**Reserved**
	3	**Is Last MB** Value / Name / Description: 1h — The current MB is the last MB in the current Slice 0h — The current MB is not the last MB in the current Slice
	2:0	**First Macroblock Bit Offset** Format: U3 This field provides the bit offset of the first macroblock in the first byte of the input bitstream.
1	31:29	**Reserved** Format: MBZ
	28:24	**Quantizer Scale Code** Format: U5 This field sets the quantizer scale code of the inverse quantizer. It remains in effect until changed by a decoded quantizer scale code in a macroblock. This field is decoded from the slice header by host software.
	23:17	**Reserved** Format: MBZ
	16:8	**Next Slice Vertical Position** Format: U9 in macroblocks This field indicates the vertical position (in macroblock units) of the first macroblock in the next slice. **Programming Notes** This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set to the height of the picture (field picture will be in height of field) (since y-direction is zero-based numbering).
	7:0	**Next Slice Horizontal Position** Format: U8 in macroblocks This field indicates the horizontal position (in macroblock units) of the first macroblock in the

MFD_MPEG2_BSD_OBJECT Inline Data Description		
		next slice. **Programming Notes** This field is primarily used for error concealment. In the case that current slice is the last slice, this field should set 0.

MPEG2

MPEG2		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	

DWord	Bit	Description
0	15:6	**Reserved** Format: MBZ
	5	**Missing EOB Error** This flag indicates missing EOB SEs coded in the bit-stream. Missing EOBs are concealed to match CBP of the error MB.
	4	**Inconsistent starting position Error - overlapping MBs** This flag indicates two slices overlapping one another by one or more MBs. Duplicate MBs decoded off the second slice shall be discarded.
	3	**Slice out-of-bound Error** This flag indicates a slice is running beyond the width of the picture. Out-of-bound MBs shall be discarded.
	2	**Premature frame end Error** This flag indicates missing slices/MBs coded in the bit-stream of a frame. One or more MBs are concealed to reach end of picture.
	1	**Inconsistent starting position Error - Missing MBs** This flag indicates one or more MBs are being concealed due to inconsistent MB starting and ending positions between slices.
	0	**MB Concealment Flag** . Each pulse from this flag indicates one MB is concealed by hardware.

MPEG4-2_Inline_DMEM

MPEG4-2_Inline_DMEM	
Project:	CHV, BSW
Source:	PRM
Size (in bits):	1600
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0	31:3	**Reserved** Format: MBZ
	2:0	**hw_used_bits** Index into the first valid bit of the starting byte of the first macroblock of the frame.
1	31:1	**Reserved** Format: MBZ
	0	**hw_short_video_header** MPEG4-2 SE: short_video_header
2	31:2	**Reserved** Format: MBZ
	1:0	**hw_shape** MPEG4-2 SE: video_object_layer_shape (rectangular support only)
3	31:16	**Reserved** Format: MBZ
	15:0	**hw_vop_time_incr_resolution_bits** MPEG4-2 SE: vop_time_increment_resolution
4	31:13	**Reserved** Format: MBZ
	12:0	**hw_width** MPEG4-2 SE: vop_width
5	31:13	**Reserved** Format: MBZ
	12:0	**hw_height**

MPEG4-2_Inline_DMEM

		MPEG4-2 SE: vop_height
6	31:1	**Reserved** Format: MBZ
	0	**hw_interlaced** MPEG4-2 SE: interlaced
7	31:1	**Reserved** Format: MBZ
	0	**hw_obmc_disable** MPEG4-2 SE: obmc_disable **Programming Notes** OBMC currently not supported; must be set to zero
8	31:1	**Reserved** Format: MBZ
	0	**hw_sprite_enable** MPEG4-2 SE: sprint_enable
9	31:6	**Reserved** Format: MBZ
	5:0	**hw_sprite_warping_points** MPEG4-2 SE: no_of_sprite_warping_points **Programming Notes** Only 0 and 1 supported
10	31:2	**Reserved** Format: MBZ
	1:0	**hw_sprite_warping_accuracy** MPEG4-2 SE: sprite_warping_accuracy **Programming Notes** Only 1/2, 1/4, 1/8 supported
11	31:4	**Reserved** Format: MBZ
	3:0	**hw_quant_precision** MPEG4-2 SE: quant_precision **Programming Notes** Must be set to 5h for ASP/SP profiles
12	31:1	**Reserved** Format: MBZ
	0	**hw_quant_type** MPEG4-2 SE: quant_type

MPEG4-2_Inline_DMEM		
13	31:1	**Reserved** Format: MBZ
	0	**hw_quarter_sample** MPEG4-2 SE: quarter_sample
14	31:1	**Reserved** Format: MBZ
	0	**hw_resync_marker_disable** MPEG4-2 SE: resync_marker_disable
15	31:1	**Reserved** Format: MBZ
	0	**hw_data_partitioned** MPEG4-2 SE: data_partitioned **Programming Notes** Data partitioning currently not supported; must be set to zero
16	31:1	**Reserved** Format: MBZ
	0	**hw_reversible_vlc** MPEG4-2 SE: reversible_vlc
17	31:7	**Reserved** Format: MBZ
	6:0	**hw_MacroBlockPerRow** Number of macroblocks per row, trunc(vop_width + 15 » 4)
18	31:7	**Reserved** Format: MBZ
	6:0	**hw_MacroBlockPerCol** Number of macroblocks per column, trunc(vop_height + 15 » 4)
19	31:15	**Reserved** Format: MBZ
	14:0	**hw_MacroBlockPerVOP** Number of macroblocks per VOP, MacroBlockPerRow * MacroBlockPerCol
20	31:4	**Reserved** Format: MBZ
	3:0	**hw_length_of_MB_number_code** Length of macroblock number code (1-14) in Table 6-27 column one of the MPEG4-2 standard specification
21	31:0	**hw_Tframe** Tframe calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification

MPEG4-2_Inline_DMEM		
22	31:0	**hw_TRD** TRD calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification
23	31:0	**hw_TRB** TRB calculation is described in section 7.7.2.2 Motion vector decoding in B-VOP in the MPEG4-2 standard specification
24	31:2	**Reserved** Format: MBZ
	1:0	**hw_coding_type** MPEG4-2 SE: vop_coding_type
25	31:1	**Reserved** Format: MBZ
	0	**hw_rounding_type** MPEG4-2 SE: rounding_type
26	31:3	**Reserved** Format: MBZ
	2:0	**hw_intra_dc_vlc_thr** MPEG4-2 SE: intra_dc_vlc_thr
27	31:1	**Reserved** Format: MBZ
	0	**hw_top_field_first** MPEG4-2 SE: top_field_first
28	31:1	**Reserved** Format: MBZ
	0	**hw_alt_vertical_scan_flag** MPEG4-2 SE: alt_vertical_scan_flag
29	31:0	**hw_warping_mv_code_du[0]** MPEG4-2 SE: warping_mv_code(du[0])
30	31:0	**hw_warping_mv_code_du[1]** MPEG4-2 SE: warping_mv_code(du[1])
31	31:0	**hw_warping_mv_code_du[2]** MPEG4-2 SE: warping_mv_code(du[2])
32	31:0	**hw_warping_mv_code_du[3]** MPEG4-2 SE: warping_mv_code(du[3])
33	31:0	**hw_warping_mv_code_dv[0]** MPEG4-2 SE: warping_mv_code(dv[0])
34	31:0	**hw_warping_mv_code_dv[1]** MPEG4-2 SE: warping_mv_code(dv[1])
35	31:0	**hw_warping_mv_code_dv[2]**

MPEG4-2_Inline_DMEM		
		MPEG4-2 SE: warping_mv_code(dv[2])
36	31:0	**hw_warping_mv_code_dv[3]** MPEG4-2 SE: warping_mv_code(dv[3])
37	31:5	**Reserved** Format: MBZ
	4:0	**hw_quant** MPEG4-2 SE: vop_quant for non-short header mode only (see **hw_263_vop_quant** for short header mode)
38	31:3	**Reserved** Format: MBZ
	2:0	**hw_fcode_forward** MPEG4-2 SE: vop_fcode_forward
39	31:3	**Reserved** Format: MBZ
	2:0	**hw_fcode_backward** MPEG4-2 SE: vop_fcode_backward
40	31:9	**Reserved** Format: MBZ
	8:0	**hw_quant_scale** MPEG4-2 SE: quant_scale
41	31:8	**Reserved** Format: MBZ
	7:0	**hw_263_temporal_reference** MPEG4-2 SE: temporal_reference (short header format only)
42	31:5	**Reserved** Format: MBZ
	4:0	**hw_263_vop_quant** MPEG4-2 SE: vop_quant (short header format only)
43	31:5	**Reserved** Format: MBZ
	4:0	**hw_263_gob_number** MPEG4-2 SE: gob_number (short header format only)
44	31:0	**hw_263_num_gobs_in_vop** Derived from Table 6-29 in the MPEG4-2 standard specification (short header format only)
45	31:0	**hw_263_num_macroblocks_in_gob** Derived from Table 6-29 in the MPEG4-2 standard specification (short header format only)
46	31:1	**Reserved** Format: MBZ

MPEG4-2_Inline_DMEM		
	0	**hw_263_gob_header_empty** MPEG4-2 SE: gob_header_empty (short header format only)
47	31:1	**Reserved** Format: MBZ
	0	**hw_263_gob_frame_id** MPEG4-2 SE: gob_frame_id (short header format only)
48	31:9	**Reserved** Format: MBZ
	8:0	**hw_263_quant_scale** MPEG4-2 SE: quant_scale (short header format only)
49	31:0	**hw_263_num_rows_in_gob** Refer to Table 6-29 in the MPEG4-2 standard specification, vop_height/(16*num_gobs_in_vop) (short header format only)

MsgDescpt31

MsgDescpt31		
Source:	Eulsa	
Size (in bits):	29	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	28:25	**Message Length** This field specifies the number of 256-bit MRF registers starting from <curr_dest> to be sent out on the request message payload. Valid value ranges from 1 to 15. A value of 0 is considered erroneous. **Value** / **Name** 1-15 / Number of MRF Registers
	24:20	**Response Length** This field indicates the number of 256-bit registers expected in the message response. The valid value ranges from 0 to 16. A value 0 indicates that the request message does not expect any response. The largest response supported is 16 GRF registers. **Value** / **Name** 0-16 / Number of Registers
	19	**Header Present** Format: / Enable If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details.
	18:0	**Function Control** This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.

Normal Media Block Message Header

MH_MB - Normal Media Block Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Project: All Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** Must be DWord aligned (Bits 1:0 MBZ) for the write form of the message.
1	31:0	**Y Offset** Project: All Format: S31 Y offset (in rows) of the upper left corner of the block into the surface.
2	31:0	**Normal Media Block Message Control** Project: All Format: MHC_MB_CONTROL [CHV, BSW] Specifies the Normal message subtype and additional input parameters.
3	31:0	**Mask** Project: All Format: U32 The Mask is ignored by the Normal Media Block message: all Dwords are always returned on reads, and always enabled to be written on writes.
4	31:0	**FFTID** Project: All Format: MHC_FFTID [CHV, BSW] Fixed Function Thread ID
5-7	95:0	**Reserved** Project: All Format: Ignore

MH_MB - Normal Media Block Message Header		
		Ignored

Normal Media Block Message Header Control

MHC_MB_CONTROL - Normal Media Block Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Project: All Format: Enumeration Specifies the interpretation of M0.3 (Pixel or Byte Mask). For the Sampler Cache Data Port, this field is ignored, behaving as if always set to NORMAL. Value / Name / Description / Project: 00h / Normal / The Block Height and Block Width fields are specified in this Dword. The Mask is ignored by a media block read message and behaves as if it is set to all ones for a media block write message. / All Others / Reserved / Reserved. / All **Programming Notes** The Media Block Read message is Normal subtype when both Sub-Register Offset and Register Pitch Control are zero. The Media Block Read message is Merged subtype when either Sub-Register Offset or Register Pitch Control are non-zero.
	29	**Reserved** Project: All Format: Ignore Ignored
	28:24	**Sub-Register Offset** Project: All Format: MBZ The sub-register offset must be 0 for Normal Media Block Read message subtype. This field is ignored (reserved) for a media block write message.
	23:22	**Reserved** Project: All Format: Ignore Ignored

MHC_MB_CONTROL - Normal Media Block Message Header Control

21:16	**Block Height** Project: All Format: U6 Height in rows of block being accessed. Range = [0,63] representing 1 to 64 rows **Restriction** If Block Width (bytes), then Maximum Block Height (rows) is constrained by (# Dwords width) * (# rows) <= 64 Dwords.
15:10	**Reserved** Project: All Format: Ignore Ignored
9:8	**Register Pitch Control** Project: All Format: MBZ The register pitch must be 0 for a Normal Media Block Read message. This field is ignored (reserved) for a media block write message.
7:6	**Reserved** Project: All Format: Ignore Ignored
5:0	**Block Width** Project: All Format: U6 Width in bytes of the block being accessed. For normal Media Block Writes, Range = [0,63] representing 1 to 64 Bytes. For normal Media Block Reads and for masked and merged Media Block messages, Range = [0,31] representing 1 to 32 Bytes. **Programming Notes** Must be DWord aligned for the write form of the message.

oMask Message Data Payload Register

MDPR_OMASK - oMask Message Data Payload Register

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:16	**oMask1** Project: All Format: U16 oMask for Pixels [15:0] of Slot 1. Not used for Slot Group HI.
	15:0	**oMask0** Project: All Format: U16 oMask for Pixels [15:0] of Slot 0. Not used for Slot Group HI.
1	31:16	**oMask3** Project: All Format: U16 oMask for Pixels [15:0] of Slot 3. Not used for Slot Group HI.
	15:0	**oMask2** Project: All Format: U16 oMask for Pixels [15:0] of Slot 2. Not used for Slot Group HI.
2	31:16	**oMask5** Project: All Format: U16 oMask for Pixels [15:0] of Slot 5. Not used for Slot Group HI.
	15:0	**oMask4** Project: All Format: U16 oMask for Pixels [15:0] of Slot 4. Not used for Slot Group HI.

MDPR_OMASK - oMask Message Data Payload Register

3	31:16	**oMask7** Project: All Format: U16 oMask for Pixels [15:0] of Slot 7. Not used for Slot Group HI.
	15:0	**oMask6** Project: All Format: U16 oMask for Pixels [15:0] of Slot 6. Not used for Slot Group HI.
4	31:16	**oMask9** Project: All Format: U16 oMask for Pixels [15:0] of Slot 9. Used only if Slot Group HI or SIMD16.
	15:0	**oMask8** Project: All Format: U16 oMask for Pixels [15:0] of Slot 8. Used only if Slot Group HI or SIMD16.
5	31:16	**oMask11** Project: All Format: U16 oMask for Pixels [15:0] of Slot 11. Used only if Slot Group HI or SIMD16.
	15:0	**oMask10** Project: All Format: U16 oMask for Pixels [15:0] of Slot 10. Used only if Slot Group HI or SIMD16.
6	31:16	**oMask13** Project: All Format: U16 oMask for Pixels [15:0] of Slot 13. Used only if Slot Group HI or SIMD16.
	15:0	**oMask12** Project: All Format: U16 oMask for Pixels [15:0] of Slot 12. Used only if Slot Group HI or SIMD16.

MDPR_OMASK - oMask Message Data Payload Register

7	31:16	**oMask15** Project: All Format: U16 oMask for Pixels [15:0] of Slot 15. Used only if Slot Group HI or SIMD16.
	15:0	**oMask14** Project: All Format: U16 oMask for Pixels [15:0] of Slot 14. Used only if Slot Group HI or SIMD16.

OM Replicated SIMD16 Render Target Data Payload

MDP_RTW_M16REP - OM Replicated SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [15:0] oMask
1.0-1.7	255:0	**RGBA** Project: All Format: MDPR_RGBA [CHV, BSW] RGBA for all slots [15:0]

OM S0A SIMD8 Render Target Data Payload

MDP_RTW_MA8 - OM S0A SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [7:0] oMask. Upper half ignored.
2.0-2.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
3.0-3.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
4.0-4.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue

MDP_RTW_MA8 - OM S0A SIMD8 Render Target Data Payload

5.0-5.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

OM S0A SIMD16 Render Target Data Payload

MDP_RTW_MA16 - OM S0A SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2816
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Source 0 Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Source 0 Alpha
2.0-2.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [15:0] oMask
3.0-3.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red

MDP_RTW_MA16 - OM S0A SIMD16 Render Target Data Payload

4.0-4.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red
5.0-5.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
6.0-6.7	255:0	**Green[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green
7.0-7.7	255:0	**Blue[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
8.0-8.7	255:0	**Blue[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
9.0-9.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
10.0-10.7	255:0	**Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Alpha

OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] oMask for slots [7:0] and [15:8]. Operation selects upper or lower half.
1.0-1.7	255:0	**Src0 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Red
2.0-2.7	255:0	**Src0 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Green
3.0-3.7	255:0	**Src0 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Blue

MDP_RTW_M8DS - OM SIMD8 Dual Source Render Target Data Payload

4.0-4.7	255:0	**Src0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Alpha
5.0-5.7	255:0	**Src1 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Red
6.0-6.7	255:0	**Src1 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Alpha

OM SIMD8 Render Target Data Payload

MDP_RTW_M8 - OM SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [7:0] oMask. Upper half ignored.
1.0-1.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

OM SIMD16 Render Target Data Payload

MDP_RTW_M16 - OM SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [15:0] oMask
1.0-1.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red
3.0-3.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
4.0-4.7	255:0	**Green[15:8]** Project: All

MDP_RTW_M16 - OM SIMD16 Render Target Data Payload

		Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green
5.0-5.7	255:0	**Blue[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
6.0-6.7	255:0	**Blue[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
7.0-7.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
8.0-8.7	255:0	**Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Alpha

Oword 1 Dual Block Data Payload

MDP_OWD1 - Oword 1 Dual Block Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword Slot0** Project: All Format: U128 Specifies the Slot 0 data
0.4-0.7	127:0	**Oword Slot1** Project: All Format: U128 Specifies the Slot 1 data

Oword 2 Block Data Payload

MDP_OW2 - Oword 2 Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword0** Project: All Format: U128 Specifies the Oword data for block element 0
0.4-0.7	127:0	**Oword1** Project: All Format: U128 Specifies the Oword data for block element 1

Oword 4 Block Data Payload

MDP_OW4 - Oword 4 Block Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[1:0]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [1:0]
1.0-1.7	255:0	**Data[3:2]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [3:2]

Oword 4 Dual Block Data Payload

MDP_OWD4 - Oword 4 Dual Block Data Payload

Project: CHV, BSW

Source: PRM

Size (in bits): 1024

Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword0 Slot0** Project: All Format: U128 Specifies the Slot 0 data for block element 0
0.4-0.7	127:0	**Oword0 Slot1** Project: All Format: U128 Specifies the Slot 1 data for block element 0
1.0-1.3	127:0	**Oword1 Slot0** Project: All Format: U128 Specifies the Slot 0 data for block element 1
1.4-1.7	127:0	**Oword1 Slot1** Project: All Format: U128 Specifies the Slot 1 data for block element 1
2.0-2.3	127:0	**Oword2 Slot0** Project: All Format: U128 Specifies the Slot 0 data for block element 2
2.4-2.7	127:0	**Oword2 Slot1** Project: All

MDP_OWD4 - Oword 4 Dual Block Data Payload

		Format: U128 Specifies the Slot 1 data for block element 2
3.0-3.3	127:0	**Oword3 Slot0** Project: All Format: U128 Specifies the Slot 0 data for block element 3
3.4-3.7	127:0	**Oword3 Slot1** Project: All Format: U128 Specifies the Slot 1 data for block element 3

Oword 8 Block Data Payload

MDP_OW8 - Oword 8 Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	1024	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.7	255:0	**Data[1:0]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [1:0]
1.0-1.7	255:0	**Data[3:2]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [3:2]
2.0-2.7	255:0	**Data[5:4]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [5:4]
3.0-3.7	255:0	**Data[7:6]** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Oword data for block elements [7:6]

Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload

MDP_A64_AOP4X2_OW2 - Oword A64 SIMD4x2 Atomic CMPWR16B Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Src0 Slot0** Format: U128 Specifies the Slot 0 Source 0 data
0.4-0.7	127:0	**Src0 Slot1** Format: U128 Specifies the Slot 1 Source 0 data
1.0-1.3	127:0	**Src1 Slot0** Format: U128 Specifies the Slot 0 Source 1 data
1.4-1.7	127:0	**Src1 Slot1** Format: U128 Specifies the Slot 1 Source 1 data

Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload

MDP_A64_AOP4X2_OW1 - Oword A64 SIMD4x2 Atomic Operation Return Data Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.3	127:0	**Oword0** Format: U128 Specifies the Slot 0 Return data
0.4-0.7	127:0	**Oword1** Format: U128 Specifies the Slot1 Return data

Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload

MDP_A64_AOP8_OW2 - Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[1:0] Src0** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [1:0] Source 0 data
1.0-1.7	255:0	**Slot[3:2] Src0** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [3:2] Source 0 data
2.0-2.7	255:0	**Slot[5:4] Src0** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [5:4] Source 0 data
3.0-3.7	255:0	**Slot[7:6] Src0** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [7:6] Source 0 data

MDP_A64_AOP8_OW2 - Oword A64 SIMD8 Atomic Operation CMPWR16B Message Data Payload

4.0-4.7	255:0	**Slot[1:0] Src1** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [1:0] Source 1 data
5.0-5.7	255:0	**Slot[3:2] Src1** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [3:2] Source 1 data
6.0-6.7	255:0	**Slot[5:4] Src1** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [5:4] Source 1 data
7.0-7.7	255:0	**Slot[7:6] Src1** Project: All Format: MDCR_OW [CHV, BSW] Specifies the Slot [7:6] Source 1 data

Oword Data Blocks Message Descriptor Control Field

MDC_DB_OW - Oword Data Blocks Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2:0	**Data Blocks** Project: All Format: Enumeration Specifies the number of Oword blocks to be read or written

Value	Name	Description	Project
00h	OW1L	1 Oword, read into or written from the low 128 bits of the destination register	All
01h	OW1U	1 Oword, read into or written from the high 128 bits of the destination register	All
02h	OW2	2 Owords	All
03h	OW4	4 Owords	All
04h	OW8	8 Owords	All
Others	Reserved	Ignored	All

Oword Data Payload Register

MDCR_OW - Oword Data Payload Register		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Oword0** Project: All Format: U128 Specifies the slot 0 data in this payload register
0.4-0.7	127:0	**Oword1** Project: All Format: U128 Specifies the slot 1 data in this payload register

Oword Dual Data Blocks Message Descriptor Control Field

MDC_DB_OWD - Oword Dual Data Blocks Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2
Default Value:	0x00000000

DWord	Bit	Description
0	1:0	**OW Dual Data Blocks** Project: All Format: Enumeration Specifies the number of Oword Blocks to be read or written

Value	Name	Description	Project
00h	OWD1	1 Hword register, 2 Owords	All
02h	OWD4	4 Hword registers, 8 Owords	All
Others	Reserved	Ignored	All

PALETTE_ENTRY

PALETTE_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	32	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	31:24	**Alpha** Format: U8 Alpha channel value for this entry in the texture color palette.
	23:16	**Red** Format: U8 Red channel value for this entry in the texture color palette.
	15:8	**Green** Format: U8 Green channel value for this entry in the texture color palette.
	7:0	**Blue** Format: U8 Blue channel value for this entry in the texture color palette.

Per Thread Scratch Space Message Header Control

MHC_PTSS - Per Thread Scratch Space Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:4	**Reserved** Project: All Format: Ignore Ignored
	3:0	**Per Thread Scratch Space** Project: All Format: U4 Specifies the amount of scratch space allowed to be used by this thread for messages in which the Binding Table Index is Stateless model, otherwise this field is ignored.The data port will use this to bounds check scratch space messages. Value range = [0,11] represents [1KB, 2MB] in powers of two. **Programming Notes** Writes out of bounds will be ignored. Reads out of bounds will return 0.

Pixel Masked Media Block Message Header

MH_MBPM - Pixel Masked Media Block Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Project: All Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** When Message Mode is set to PIXEL_MASK, this field must be a multiple of 32.
1	31:0	**Y Offset** Project: All Format: S31 Y offset (in rows) of the upper left corner of the block into the surface. **Programming Notes** When Message Mode is set to PIXEL_MASK, this field must be a multiple of 4.
2	31:0	**Media Block Message Control** Project: All Format: MHC_MBPM_CONTROL Specifies the message subtype is Pixel Masked.
3	31:0	**Pixel Mask** Project: All Format: U32 Specifies the Pixel Mask for writes when Message Mode field is PIXEL_MASK. **Programming Notes** The Pixel Mask applies to the 2x2 square tiles (UL, UR, LL, LR), which themselves tiled (UL, UR, LL, LR) and then repeated on the right for the remaining 16-bits to cover a 4 row 8 column area.
4	31:0	**FFTID** Project: All Format: MHC_FFTID [CHV, BSW] Fixed Function Thread ID

MH_MBPM - Pixel Masked Media Block Message Header		
5-7	95:0	**Reserved** Project: All Format: Ignore Ignored

Pixel Masked Media Block Message Header Control

MHC_MBPM_CONTROL - Pixel Masked Media Block Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:30	**Message Mode** Project: All Format: Enumeration Specifies the Media Block Write Message subtype is Pixel Masked. Value / Name / Description / Project: 01h / PIXEL_MASK / Use the Pixel Mask in the Message Header. The Block Height and Block Width are ignored and behave as if they are set to 4 rows and 32 bytes, respectively. / All Others / Reserved / Reserved. / All
	29	**Reserved** Project: All Format: Ignore Ignored
	28:24	**Sub-Register Offset** Project: All Format: U5 This field is ignored (reserved) for a media block write message.
	23:22	**Reserved** Project: All Format: Ignore Ignored
	21:16	**Block Height** Project: All Format: U6 This field is ignored (reserved) for a Pixel Masked media block write message.
	15:10	**Reserved**

MHC_MBPM_CONTROL - Pixel Masked Media Block Message Header Control

		Project: All Format: Ignore Ignored
	9:8	**Register Pitch Control** Project: All Format: U2 This field is ignored (reserved) for a media block write message.
	7:6	**Reserved** Project: All Format: Ignore Ignored
	5:0	**Block Width** Project: All Format: U6 This field is ignored (reserved) for a Pixel Masked media block write message.

Pixel Sample Mask Message Header Control

MHC_PSM - Pixel Sample Mask Message Header Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x0000FFFF

DWord	Bit	Description
0	31:16	**Reserved** Format: Ignore Ignored
	15:0	**Pixel Sample Mask** Default Value: 0FFFFh Default Format: U16 SIMD16 and SIMD8 messages. All 16 bits are used for SIMD16. For untyped SIMD8 messages, the low 8 bits of field are used. If the header is not delivered, this field defaults to all ones. This field is ignored for SIMD4x2 messages.

Pixel Sample Mask Render Target Message Header Control

MHC_RT_PSM - Pixel Sample Mask Render Target Message Header Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Dispatched Pixel/Sample Enables** Project: All Format: U16 One bit per pixel (or sample within pixel) indicating which pixels/samples were originally enabled when the thread was dispatched. The Dispatched Pixel/Sample Enables must be unmodified from the ones sent when the pixel shader thread was initiated. If the Dispatched Pixel/Sample Enables are modified, behavior is undefined. **Programming Notes** When operating in PER_SAMPLE mode these bits correspond to samples, not pixels. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. Note that in NUMSAMPLES_1 mode, a pixel and sample are synonomous. When operating in PER_PIXEL mode, this field is ignored, and instead the SampleEnableMask (obtained via bypass) are used to clear the Depth Scoreboard.
	15:0	**Pixel/Sample Enables** Project: All Format: U16 Specifies which pixels/samples are still lit based on kill instruction activity in the pixel shader. This mask is AND'd with the Dispatched Pixel/Sample Enables mask, and that is used to control actual accesses to the color buffer. Pixels/samples will be dropped on masked writes, and the GRF is not modified for masked reads. **Programming Notes** When operating in PER_SAMPLE mode these bits correspond to samples, not pixels, as the PS is run per-sample. Each subspan slot (4 bits) corresponds to a specific sample location for the subspan. When operating in PER_PIXEL mode, these bits still correspond to pixels, as the PS is run per-pixel. Each pixels mask bit is replicated according to Number of Multisamples and combined with other masks to control writes to the multisample locations.

Power Clock State Format

Power Clock State Format

Project:	CHV, BSW
Source:	RenderCS
Size (in bits):	31
Default Value:	0x00000288

Known Uses

- R_PWR_CLK_STATE - Render Power Clock State Register
- PM_PWR_CLK_STATE - PM Power Clock State Request (Intended, in GT/GTI space, not yet in use)
- PM_PWR_CLK_STATE (Intended, in GT/GTI space, not yet in use)

DWord	Bit	Description
0 **Project:** CHV, BSW	30:19	**RSVD** Project: CHV, BSW Access: RO Format: MBZ Reserved (CSunit implements full 32b storage)
	18	**SCountEn** Project: CHV, BSW Access: R/W **Programming Notes** Not supported in CHV, BSW, . Must be zero (MBZ).
	17:15	**SliceCount** Project: CHV, BSW Access: R/W **Programming Notes** Not supported in CHV, BSW, . Must be zero (MBZ).
	14:13	**RSVD** Project: CHV, BSW Access: RO Reserved (CSunit implements full 32b storage)
	12	**Spare** Project: CHV, BSW

Power Clock State Format

	Access:	R/W
	Format:	MBZ

Spare bit for CHV, BSW

11 Reserved

Project:	CHV, BSW
Access:	R/W

10:8 Reserved

Project:	CHV, BSW
Access:	R/W

7:4 EUmax

Project:	CHV, BSW
Access:	R/W

Maximum number of EUs to power (per subslice if multiple subslices enabled). To specify an exact number of subslices, set EUmax equal to EUmin.

Value	Name	Description
0010b		2 EUs
0100b		4 EUs
0110b		6 EUs
1000b	**[Default]**	8 EUs

Programming Notes
EUmin and EUmax need to be even and odd numbers are illegal.

3:0 EUmin

Project:	CHV, BSW
Access:	R/W

Minimum number of EUs to power (per subslice if multiple subslices enabled).

Value	Name	Description
0010b		2 EUs
0100b		4 EUs
0110b		6 EUs
1000b	**[Default]**	8 EUs (minimum for GPGPU workloads)

Programming Notes
EUmin and EUmax need to be even and odd numbers are illegal.

PPHWSP_LAYOUT

PPHWSP_LAYOUT - PPHWSP_LAYOUT		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	1089	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..3	31:0	**Reserved**
4	31:0	**Ring Head Pointer Storage** The contents of the Ring Buffer Head Pointer register (register DWord 1) are written to this location either as result of an MI_REPORT_HEAD instruction or as the result of an “automatic report” (see RINGBUF registers).
5..15	31:0	**Reserved**
16	0	**Cumulative Context Run Time** This has the cumulative run time of the context on HW. HW reports CTX_TIMESTAMP to this location on a context switch.
17	31:1	**Reserved**
	0	**Reserved** Project: CHV, BSW
18..19	63:0	**Preempt Request Received Timestamp** TIMESTAMP register sampled on preemption request is reported.
20..21	63:0	**Context Restore Complete Timestamp** TIMESTAMP register sampled on context restore complete is reported.
22..23	63:0	**Context Save Finished Timestamp** TIMESTAMP register sampled on context save completion is reported.
24..27	127:0	**MI_SEMAPHORE_WAIT** MI_SEMAPHORE_WAIT command on which the context got switched out due to semaphore wait. This field is only valid and must be looked at when the context switch reason in context status buffer is stated as “Wait on Semaphore”.
28..31	127:0	**Reserved**
32..33 **Project:**	63:0	**Reserved** Project:
1020-34	31:0	**Reserved**

Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload

MDP_A64_AOP4X2_QW2 - Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Src0 Slot0** Format: U64 Specifies the Slot 0 Source 0 data
0.2-0.3	63:0	**Reserved** Format: Ignore Ignored
0.4-0.5	63:0	**Src0 Slot1** Format: U64 Specifies the Slot 1 Source 0 data
0.6-0.7	63:0	**Reserved** Format: Ignore Ignored
1.0-1.1	63:0	**Src1 Slot0** Format: U64 Specifies the Slot 0 Source 1 data
1.2-1.3	63:0	**Reserved** Format: Ignore Ignored
1.4-1.5	63:0	**Src1 Slot1** Format: U64 Specifies the Slot 1 Source 1 data
1.6-1.7	63:0	**Reserved**

MDP_A64_AOP4X2_QW2 - Qword A64 SIMD4x2 Atomic CMPWR Message Data Payload

		Format:	Ignore
		Ignored	

Qword Data Payload Register

MDCR_QW - Qword Data Payload Register

Project: CHV, BSW
Source: PRM
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Qword0** Project: All Format: U64 Specifies the slot 0 data in this payload register
0.2-0.3	63:0	**Qword1** Project: All Format: U64 Specifies the slot 1 data in this payload register
0.4-0.5	63:0	**Qword2** Project: All Format: U64 Specifies the slot 2 data in this payload register
0.6-0.7	63:0	**Qword3** Project: All Format: U64 Specifies the slot 3 data in this payload register

Qword SIMD4x2 Atomic CMPWR8B Message Data Payload

MDP_AOP4X2_QW2 - Qword SIMD4x2 Atomic CMPWR8B Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**Src0 Slot0** Format: U64 Specifies the Slot 0 Source 0 data
2-3	63:0	**Src1 Slot0** Format: U64 Specifies the Slot 0 Source 1 data
4-5	63:0	**Src0 Slot1** Format: U64 Specifies the Slot 1 Source 0 data
6-7	63:0	**Src1 Slot1** Format: U64 Specifies the Slot 1 Source 1 data

Qword SIMD4x2 Atomic Operation Message Data Payload

MDP_AOP4X2_QW1 - Qword SIMD4x2 Atomic Operation Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**Qword0** Format: U64|S63 Specifies the Slot 0 Source or Return data
2-3	63:0	**Reserved** Format: Ignore Ignored
4-5	63:0	**Qword1** Format: U64|S63 Specifies the Slot 1 Source or Return data
6-7	63:0	**Reserved** Format: Ignore Ignored

Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload

MDP_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR8B Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Src0[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Slot [7:0] Source 0 data
1.0-1.7	255:0	**Slot[7:0] Src0[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Slot [7:0] Source 0 data
2.0-2.7	255:0	**Slot[7:0] Src1[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Slot [7:0] Source 1 data
3.0-3.7	255:0	**Slot[7:0] Src1[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Slot [7:0] Source 1 data

Qword SIMD8 Atomic Operation CMPWR Message Data Payload

MDP_A64_AOP8_QW2 - Qword SIMD8 Atomic Operation CMPWR Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[3:0] Src0** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [3:0] Source 0 data
1.0-1.7	255:0	**Slot[7:4] Src0** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [7:4] Source 0 data
2.0-2.7	255:0	**Slot[3:0] Src1** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [3:0] Source 1 data
3.0-3.7	255:0	**Slot[7:4] Src1** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [7:4] Source 1 data

Qword SIMD8 Atomic Operation Return Data Message Data Payload

MDP_AOP8_QW1 - Qword SIMD8 Atomic Operation Return Data Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Qword[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Slot [7:0] Return data
1.0-1.7	255:0	**Slot[7:0] Qword[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Slot [7:0] Return data

Qword SIMD8 Data Payload

MDP_QW_SIMD8 - Qword SIMD8 Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[3:0]** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [3:0] data
1.0-1.7	255:0	**Data[7:4]** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [7:4] data

Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload

MDP_AOP16_QW2 - Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Src0[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Source 0 data for Slot [7:0]
1.0-1.7	255:0	**Slot[15:8] Src0[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits Source 0 data for Slot [15:8]
2.0-2.7	255:0	**Slot[7:0] Src0[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Source 0 data for Slot [7:0]
3.0-3.7	255:0	**Slot[15:8] Src0[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits Source 0 data for Slot [15:8]

MDP_AOP16_QW2 - Qword SIMD16 Atomic Operation CMPWR8B Message Data Payload

4.0-4.7	255:0	**Slot[7:0] Src1[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Source 1 data for Slot [7:0]
5.0-5.7	255:0	**Slot[15:8] Src1[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits Source 1 data for Slot [15:8]
6.0-6.7	255:0	**Slot[7:0] Src1[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Source 1 data for Slot [7:0]
7.0-7.7	255:0	**Slot[15:8] Src1[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits Source 1 data for Slot [15:8]

Qword SIMD16 Atomic Operation Return Data Message Data Payload

MDP_AOP16_QW1 - Qword SIMD16 Atomic Operation Return Data Message Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Slot[7:0] Qword[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Return data for Slot [7:0]
1.0-1.7	255:0	**Slot[15:8] Qword[31:0]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the lower 32-bits of Return data for Slot [15:8]
2.0-2.7	255:0	**Slot[7:0] Qword[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Return data for Slot [7:0]
3.0-3.7	255:0	**Slot[15:8] Qword[63:32]** Project: All Format: MDCR_DW [CHV, BSW] Specifies the upper 32-bits of Return data for Slot [15:8]

Qword SIMD16 Data Payload

MDP_QW_SIMD16 - Qword SIMD16 Data Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Data[3:0]** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [3:0] data
1.0-1.7	255:0	**Data[7:4]** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [7:4] data
2.0-2.7	255:0	**qw11_qw8** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [11:8] data
3.0-3.7	255:0	**qw15_qw12** Project: All Format: MDCR_QW [CHV, BSW] Specifies the Slot [15:12] data

Read-Only Data Port Message Types

MT_DP_RO - Read-Only Data Port Message Types

Project: CHV, BSW
Source: Read-Only DataPort
Size (in bits): 5
Default Value: 0x00000000

Lists all the Message Types in a Read-Only Data Port Message Descriptor [18:14]. Read operations from the Constant Cache and Sampler Cache are encoded in the Read-Only Data Port. Many of the operations are also implemented in Data Port 0, and those operations use the same Message Header.

DWord	Bit	Description
0	4	**Reserved** Format: MBZ Ignored
	3:0	**Message Type** Format: Enumeration Specifies type of message

Value	Name	Description	Project
00h	MT_CC_OWB **[Default]**	Oword Block Read Constant Cache message	
01h	MT_CC_OWUB	Unaligned Oword Block Read Constant Cache message	
02h	MT_CC_OWDB	Oword Dual Block Read Constant Cache message	
03h	MT_CC_DWS	Dword Scattered Read Constant Cache message	
04h	MT_SC_OWUB	Unaligned Oword Block Read Sampler Cache message	
05h	MT_SC_MB	Media Block Read Sampler Cache message	
06h	MT_RSI	Read Surface Info message	
Others	Reserved	Ignored	

Read Surface Info 32-Bit Address Payload

MAP32B_RSI - Read Surface Info 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**U** Project: All Format: U32 Specifies the U channel address offset.
0.1	31:0	**V** Project: All Format: U32 Specifies the V channel address offset.
0.2	31:0	**R** Project: All Format: U32 Specifies the R channel address offset.
0.3	31:0	**LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD.
0.4-0.7	127:0	**Reserved** Project: All Format: Ignore Ignored

Read Surface Info Data Payload

MDP_RSI - Read Surface Info Data Payload

Project: CHV, BSW

Source: PRM

Size (in bits): 512

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.5	191:0	**Reserved** Project: All Format: Ignore Ignored
0.6-0.7	63:0	**Instruction Base Address** Project: All Format: GraphicsAddress[63:0] Instruction Base Address from STATE_BASE_ADDRESS, extended to 64-bit format. **Programming Notes** The 48-bit address is returned in a 64-bit address in canonical form.
1.0	31:0	**Width** Project: All Format: U32 Surface Width generally computed from RENDER_SURFACE_STATE Width (stored as width minus 1). The value is 0 for NULL surface, and in all other cases (Width+1) » LOD. Surface Width from RENDER_SURFACE_STATE (U14), zero extended to 32 bits.
1.1	31:0	**Height** Project: All Format: U32 Surface Height, generally computed from RENDER_SURFACE_STATE Height (stored as height minus 1). The value for a 1D array is RENDER_SUFACE_STATE's (Depth + 1). The value for 1D non-array, BUFFER, and NULL surface is 0. In all other case, the value is (Height + 1) » LOD.
1.2	31:0	**Depth** Project: All Format: U32 Surface Depth, generally computed from RENDER_SURFACE_STATE Depth (which is stored depth minus 1). If 2D Array or Cube Array surface, value is the (Depth+1). If 3D surface, value is

MDP_RSI - Read Surface Info Data Payload

(Depth+1) » LOD. In all other case, the value is 0.

1.3 31:0 **MIP Count**

Project:	All
Format:	U32

MIP Count from RENDER_SURFACE_STATE, range [0, 14], zero extended to 32 bits.

1.4 31:0 **Surface Type**

Project:	All
Format:	U32

Surface Type from RENDER_SURFACE_STATE, zero extended to 32 bits

Value	Name	Description	Project
0h	SURFTYPE_1D	1-dimensional map or array of maps	All
1h	SURFTYPE_2D	2-dimensional map or array of maps	All
2h	SURFTYPE_3D	3-dimensional map (volumetric) of maps	All
3h	SURFTYPE_CUBE	Cube map or array of cube maps	All
4h	SURFTYPE_BUFFER	Element in a buffer	All
5h	SURFTYPE_STRBUF	Structured buffer surface	All
7h	SURTYPE_NULL	Null surface	All
Others	Reserved	Reserved	All

1.5 31:0 **Surface Format**

Project:	All
Format:	U32

Surface Format from RENDER_SURFACE_STATE (U9), zero extended to 32 bits.

1.6-1.7 63:0 **Reserved**

Project:	All
Format:	Ignore

Ignored

RENDER_SURFACE_STATE

RENDER_SURFACE_STATE

Project:	CHV, BSW
Source:	PRM
Exists If:	//[MessageType] != 'Sample_8x8'
Size (in bits):	480
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the normal surface state used by all messages that use SURFACE_STATE except those that use MEDIA_SURFACE_STATE.

DWord	Bit	Description
0	31:29	**Surface Type** This field defines the type of the surface.

Value	Name	Description
0h	SURFTYPE_1D	Defines a 1-dimensional map or array of maps
1h	SURFTYPE_2D	Defines a 2-dimensional map or array of maps
2h	SURFTYPE_3D	Defines a 3-dimensional (volumetric) map
3h	SURFTYPE_CUBE	Defines a cube map or array of cube maps
4h	SURFTYPE_BUFFER	Defines an element in a buffer
5h	SURFTYPE_STRBUF	Defines a structured buffer surface
6h	Reserved	
7h	SURFTYPE_NULL	Defines a null surface

Programming Notes

A null surface is used in instances where an actual surface is not bound. When a write message is generated to a null surface, no actual surface is written to. When a read message (including any sampling engine message) is generated to a null surface, the result is all zeros. Note that a null surface type is allowed to be used with all messages, even if it is not specificially indicated as supported. All of the remaining fields in surface state are ignored for null surfaces, with the following exceptions:

- **Width, Height, Depth, LOD,** and **Render Target View Extent** fields must match the depth buffer's corresponding state for all render target surfaces, including null.

All sampling engine and data port messages support null surfaces with the above behavior, even if not mentioned as specifically supported, except for the following:

- Data Port Media Block Read/Write messages
- Data Port Transpose Read message
- The **Surface Type** of a surface used as a render target (accessed via the Data Port's

RENDER_SURFACE_STATE

Render Target Write message) must be the same as the **Surface Type** of all other render targets and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless either the depth buffer or render targets are SURFTYPE_NULL.

For sampling using the 3D sampler, if the Surface Type is programmed to SURFTYPE_NULL, the Surface Format must be a supported surface format for the 3D sampler.

28 **Surface Array**

Format:	Enable

This field, if enabled, indicates that the surface is an array.

Programming Notes

If this field is *enabled*, the **Surface Type** must be SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE.
If this field is *disabled* and **Surface Type** is SURFTYPE_1D, SURFTYPE_2D, or SURFTYPE_CUBE, the **Depth** field must be set to zero.

27 **Reserved**

Project:	CHV, BSW
Format:	Enable

26:18 **Surface Format**

Format:	SURFACE_FORMAT [CHV, BSW]

This field specifies the format of the surface or element within this surface. This field is ignored for all data port messages other than the render target message and streamed vertex buffer write message. Some forms of the media block messages use the surface format.

17:16 **Surface Vertical Alignment**

Description	Project
For Sampling Engine and Render Target Surfaces: This field specifies the vertical alignment requirement in elements for the surface. Refer to the "Memory Data Formats" chapter for details on how this field changes the layout of the surface in memory. An *element* is defined as a pixel in uncompresed surface formats, and as a compression block in compressed surface formats. For MSFMT_DEPTH_STENCIL type multisampled surfaces, an element is a sample.	
This field applies to surface formats other than compressed formats.	CHV, BSW
For other surfaces: This field is ignored.	

Value	Name	Description
0h	Reserved	Reserved
1h	VALIGN 4	Vertical alignment factor j = 4
2h	VALIGN 8	Vertical alignment factor j = 8

RENDER_SURFACE_STATE

3h	VALIGN 16	Vertical alignment factor j = 16

Programming Notes
This field is intended to be set to VALIGN_4 if the surface was rendered as a depth buffer, for a multisampled (4x) render target, or for a multisampled (8x) render target, since these surfaces support only alignment of 4. Use of VALIGN_4 for other surfaces is supported, but increases memory usage.
This field is intended to be set to VALIGN_8 only if the surface was rendered as a stencil buffer, since stencil buffer surfaces support only alignment of 8. If set to VALIGN_8, Surface Format must be R8_UINT.
For uncompressed surfaces, the units of "j" are rows of pixels on the physical surface. For compressed texture formats, the units of "j" are in compression blocks, thus each increment in "j" is equal to h pixels, where h is the height of the compression block in pixels.

15:14 **Surface Horizontal Alignment**

Description	Project
For Sampling Engine and Render Target Surfaces: This field specifies the horizontal alignment requirement for the surface. This field is ignored when Tiled Resource Mode is not TRMODE_NONE (i.e. Tiled Resources are enabled). See the "Surface Layout and Tiling" section under Common Surface Formats for the table of alignment values for Tile Resrouces.	
This field applies to surface formats other than compressed formats.	CHV, BSW
For other surfaces: This field is ignored.	

Value	Name	Description
0h	Reserved	Reserved
1h	HALIGN 4	Horizontal alignment factor j = 4
2h	HALIGN 8	Horizontal alignment factor j = 8
3h	HALIGN 16	Horizontal alignment factor j = 16

Programming Notes
This field is intended to be set to HALIGN_8 only if the surface was rendered as a depth buffer with Z16 format or a stencil buffer. In this case it must be set to HALIGN_8 since these surfaces support only alignment of 8. For Z32 formats it must be set ot HALIGN_4. Use of HALIGN_8 for other surfaces is supported, but increases memory usage. For uncompressed surfaces, the units of "i" are pixels on the physical surface. For compressed texture formats, the units of "i" are in compression blocks, thus each increment in "i" is equal to w pixels, where w is the width of the compression block in pixels.
When Auxiliary Surface Mode is set to AUX_CCS_D or AUX_CCS_E, HALIGN 16 must be used.

13:12 **Tile Mode**

This field specifies the type of memory tiling (Linear, WMajor, XMajor, or YMajor) employed to

RENDER_SURFACE_STATE

tile this surface. See *Memory Interface Functions* for details on memory tiling and restrictions.

Value	Name	Description	Project
0h	LINEAR	Linear mode (no tiling)	
1h	WMAJOR	W major tiling	CHV, BSW
2h	XMAJOR	X major tiling	
3h	YMAJOR	Y major tiling	

Programming Notes

- Refer to *Memory Data Formats* for restrictions on *TileMode* direction for the various buffer types. (Of particular interest is the fact that YMAJOR tiling is not supported for display/overlay buffers).
- The corresponding cache(s) must be invalidated before a previously accessed surface is accessed again with an altered state of this field.
- Use of WMAJOR is valid only for sampling engine, Data Cache Data Port and render target surfaces and **Surface Format** must be R8_UINT. Vertical Line Stride must be zero. In addition to W tiling, this mode implies that the surface is stored as a stencil buffer. Refer to *Memory Data Formats* section for details on stencil buffer surface layout.
- Linear surfaces can be mapped to Main Memory (uncached) or System Memory (cacheable, snooped). Tiled (X/Y/W) surfaces can only be mapped to Main Memory.
- If **Surface Type** is SURFTYPE_BUFFER, this field must be TILEMODE_LINEAR
- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be YMAJOR.

If **Surface Format** is ASTC*, this field must be TILEMODE_YMAJOR.

11 **Vertical Line Stride**

Format:	U1 In lines to skip between logically adjacent lines

For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Cache Data Port: Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.

For Other Surfaces: Vertical Line Stride must be zero.

Programming Notes

This bit must not be set if the surface format is a compressed type (BCn*, FXT1, ETC*, EAC*, ASTC*).

This bit must not be set if the **Auxiliary Surface Mode** is not AUX_NONE.

If this bit is set on a sampling engine surface, the mip mode filter must be set to MIPFILTER_NONE and the min and mag mode filter cannot be set to MAPFILTER_FLEXIBLE.

Workaround	Project
All surfaces used by the sampler between sampler cache invalidates must have the same setting of this field in both RENDER_SURFACE_STATE and MEDIA_SURFACE_STATE.	CHV, BSW

RENDER_SURFACE_STATE

10 Vertical Line Stride Offset

Format:	U1 In lines of initial offset (when Vertical Line Stride == 1)

For 2D Non-Array Surfaces accessed via the Sampling Engine or Data Cache Data Port:
Specifies the offset of the initial line from the beginning of the buffer. Ignored when Vertical **Line Stride** is 0.
For Other Surfaces:
Vertical Line Stride Offset must be zero.

9 Sampler L2 Bypass Mode Disable

Format:	Disable

This field allows the Sampler L2 bypass mode to be disabled for the surface. If enabled, Sampler can still disable the L2 bypass as needed.

Programming Notes
This bit must be set for the following surface types: BC2_UNORM BC3_UNORM BC5_UNORM BC5_SNORM BC7_UNORM

8 Render Cache Read Write Mode
For Surfaces accessed via the Data Port to Render Cache:
This field specifies the way Render Cache treats a write request. If unset, Render Cache allocates a write-only cache line for a write miss. If set, Render Cache allocates a read-write cache line for a write miss.
For Surfaces accessed via the Sampling Engine or Data Port to Texture Cache or Data Cache:
This field is reserved : MBZ

Value	Name	Description
0h	Write-Only Cache	Allocating write-only cache for a write miss
1h	Read-Write Cache	Allocating read-write cache for a write miss

Programming Notes
This field is provided for performance optimization for Render Cache read/write accesses (from Gen4 EU's point of view).

7:6 Media Boundary Pixel Mode
For 2D Non-Array Surfaces accessed via the Data Port Media Block Read Message or Data Port Transpose Read message:
This field enables control of which rows are returned on vertical out-of-bounds reads using the Data Port Media Block Read Message or Data Port Transpose Read message. In the description below, frame mode refers to **Vertical Line Stride** = 0, field mode is **Vertical Line Stride** = 1 in which only the even or odd rows are addressable. The frame refers to the entire surface, while the field refers only to the even or odd rows within the surface.
For Other Surfaces:
Reserved : MBZ

Value	Name	Description
0h	NORMAL_MODE	The row returned on an out-of-bound access is the closest row

RENDER_SURFACE_STATE

		in the frame or field. Rows from the opposite field are never returned.
1h	Reserved	
2h	PROGRESSIVE_FRAME	The row returned on an out-of-bound access is the closest row in the frame, even if in field mode.
3h	INTERLACED_FRAME	In field mode, the row returned on an out-of-bound access is the closest row in the field. In frame mode, even out-of-bound rows return the nearest even row while odd out-of-bound rows return the nearest odd row.

5 **Cube Face Enable - Negative X**

Exists If:	[Surface Type] == 'SURFTYPE_CUBE'
Format:	Enable

For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine: This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).

4 **Cube Face Enable - Positive X**

Exists If:	[Surface Type] == 'SURFTYPE_CUBE'
Format:	Enable

For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine: This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).

3 **Cube Face Enable - Negative Y**

Exists If:	[Surface Type] == 'SURFTYPE_CUBE'
Format:	Enable

For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine: This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided.

Programming Notes

RENDER_SURFACE_STATE

		When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	2	**Cube Face Enable - Positive Y** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	1	**Cube Face Enable - Negative Z** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
	5:0	**Reserved** Exists If: [Surface Type] != 'SURFTYPE_CUBE' Format: MBZ
	0	**Cube Face Enable - Positive Z** Exists If: [Surface Type] == 'SURFTYPE_CUBE' Format: Enable **For SURFTYPE_CUBE Surfaces accessed via the Sampling Engine:** This field enable the individual face of a cube map. Enabling a face indicates that the face is present in the cube map, while disabling it indicates that that face is represented by the texture map's border color. Refer to Memory Data Formats for the correlation between faces and the cube map memory layout. Note that storage for disabled faces must be provided. **Programming Notes** When TEXCOORDMODE_CLAMP is used when accessing a cube map, this field must be programmed to 1b (face enabled).
1	31	**Use Global Clear Value** Project: CHV, BSW

RENDER_SURFACE_STATE

Format:	Enable

Specifies that the global non 0/1 clear value should be used for clearing the render target.

Value	Name	Description
0h		Use 0/1 clear value programmed in surface state
1h		Use non-pipelined global non 0/1 clear values

Programming Notes	Project
If 3DSTATE_MULTISAMPLE::**Number of Multisamples** is not NUMSAMPLES_1, this field must be set to 0.	CHV, BSW

30:24 **Memory Object Control State**

Project:	All
Format:	MEMORY_OBJECT_CONTROL_STATE

Specifies the memory object control state for this surface and the associated Auxiliary surface (if any).

23:19 **Base Mip Level**

Project:	All
Format:	U4.1

Range: [0.0, 14.0]
Specifies which mip level is considered the "base" level when determining mag-vs-min filter and selecting the "base" mip level.

Programming Notes
This field also exists in SAMPLER_STATE. If both fields are zero, the Base Mip Level is zero. If one is nonzero, Base Mip Level is the nonzero field. It is illegal to have both Base Mip Level fields nonzero.

18 **Reserved**

Project:	
Format:	MBZ

17 **Reserved**

Project:	CHV, BSW
Format:	MBZ

16:15 **Reserved**

Format:	MBZ

14:0 **Surface QPitch**

Format:	QPitch[16:2]

RENDER_SURFACE_STATE

Description	Project
This field specifies the distance in rows between array slices. It is used only in the following cases: • **Surface Array** is enabled OR • **Number of Mulitsamples** is not NUMSAMPLES_1 and **Multisampled Surface Storage Format** set to MSFMT_MSS OR • **Surface Type** is SURFTYPE_CUBE	CHV, BSW

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

Programming Notes	Project
This field must be set to an integer multiple of the **Surface Vertical Alignment**. For compressed textures (BC*, FXT1, ETC*, EAC*, ASTC Surface Formats), this field is in units of rows in the uncompressed surface, and must be set to an integer multiple of the vertical alignment parameter "j" defined in the *Common Surface Formats* section.	CHV, BSW
Software must ensure that this field is set to a value sufficiently large such that the array slices in the surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored in memory.	

2 | 31:30 **Reserved**

Format:	MBZ

29:16 **Height**

Format:	U14-1

This field specifies the height of the surface, minus 1. If the surface is MIP-mapped, this field contains the height of the base MIP level. For buffers, this field specifies a portion of the buffer size.

Value	Name	Description	Project	Exists If
[0,0]		must be zero		[Surface Type] == 'SURFTYPE_1D'
[0,16383]		height of surface - 1 (y/v dimension)		[SurfaceType] == 'SURFTYPE_2D'
[0,2047]		height of surface - 1 (y/v dimension)	CHV, BSW	[SurfaceType] == 'SURFTYPE_3D'
[0,16383]		height of surface - 1 (y/v dimension)		[SurfaceType] == 'SURFTYPE_CUBE'
[0,16383]		contains bits [20:7] of the number of entries in the buffer - 1		([SurfaceType] == 'SURFTYPE_BUFFER') || ([SurfaceType] == 'SURFTYPE_STRBUF')

Programming Notes	Project
For typed buffer and structured buffer surfaces, the number of entries in the buffer	

RENDER_SURFACE_STATE

		ranges from 1 to 2^{27}. For raw buffer surfaces, the number of entries in the buffer is the number of bytes which can range from 1 to 2^{30}. After subtracting one from the number of entries, software must place the fields of the resulting 27-bit value into the **Height, Width**, and **Depth** fields as indicated, right-justified in each field. Unused upper bits must be set to zero.	
		If **Vertical Line Stride** is 1, this field indicates the height of the field, not the height of the frame	
		The **Height** of a render target must be the same as the **Height** of the other render targets and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped).	
		If this surface in memory is accessed with Vertical Line Stride set to both 0 and 1, this field must be an even value when Vertical Line Stride is 0.	
		If Media Pixel Boundary Mode is not set to NORMAL_MODE, this field must be an even value.	
		If Surface Format is PLANAR*, this field must be a multiple of 4	CHV, BSW
	15:14	**Reserved**	
		Format:	MBZ
	13:0	**Width**	
		Format:	U14-1

Description	Project
This field specifies the width of the surface, minus 1. If the surface is MIP-mapped, this field specifies the width of the base MIP level. The width is specified in units of pixels or texels. For buffers, this field specifies a portion of the buffer size.	
For surfaces accessed with the Media Block Read/Write message, this field is in units of DWords.	CHV, BSW
For surfaces accessed with the Transpose Read Message, this field is in units of DWords.	CHV, BSW

Value	Name	Description	Project	Exists If
[0,16383]		width of surface - 1 (x/u dimension)		[SurfaceType] == 'SURFTYPE_1D'
[0,16383]		width of surface - 1 (x/u dimension)		[SurfaceType] == 'SURFTYPE_2D'
[0,2047]		width of surface - 1 (x/u dimension)	CHV, BSW	[SurfaceType] == 'SURFTYPE_3D'
[0,16383]		width of surface - 1 (x/u dimension)		[SurfaceType] == 'SURFTYPE_CUBE'
[0,127]		contains bits [6:0] of the		([SurfaceType] == 'SURFTYPE_BUFFER')

RENDER_SURFACE_STATE

		number of entries in the buffer - 1		|| ([SurfaceType] == 'SURFTYPE_STRBUF')

Programming Notes	Project
• For surface types other than SURFTYPE_BUFFER or STRBUF The Width specified by this field must be less than or equal to the surface pitch (specified in bytes via the Surface Pitch field). • For cube maps, Width must be set equal to the Height. • For MONO8 textures, Width must be a multiple of 32 texels. • The **Width** of a render target must be the same as the **Width** of the other render target(s) and the depth buffer (defined in 3DSTATE_DEPTH_BUFFER), unless **Surface Type** is SURFTYPE_1D or SURFTYPE_2D with **Depth** = 0 (non-array) and **LOD** = 0 (non-mip mapped). • The **Width** of a render target with YUV surface format must be a multiple of 2. • For SURFTYPE_BUFFER: The low two bits of this field must be 11 if the Surface Format is RAW (the size of the buffer must be a multiple of 4 bytes).	
If **Surface Format** is PLANAR*, this field must be a multiple of 4	CHV, BSW

3 | 31:21 **Depth**

Format:	U11-1

This field specifies the total number of levels, minus 1, for a volume texture or the number of array elements, minus 1, allowed to be accessed starting at the **Minimum Array Element** for arrayed surfaces. If the volume texture is MIP-mapped, this field specifies the depth of the base MIP level. For buffers, this field specifies a portion of the buffer size.

Value	Name	Description	Project	Exists If
[0,2047]		number of array elements - 1		[SurfaceType] == 'SURFTYPE_1D'
[0,2047]		number of array elements - 1		[SurfaceType] == 'SURFTYPE_2D'
[0,2047]		depth of surface - 1 (z/r dimension)		[SurfaceType] == 'SURFTYPE_3D'
[0,340]		number of array elements - 1 [see programming notes for range]		[SurfaceType] == 'SURFTYPE_CUBE'
[0,1023]		contains bits [30:21] of the number of entries in the buffer - 1	CHV, BSW	([SurfaceType] == 'SURFTYPE_BUFFER') AND ([SurfaceFormat] == 'RAW')
[0,63]		contains bits [26:21] of the number of entries in the buffer - 1	CHV, BSW	([SurfaceType] == 'SURFTYPE_BUFFER') AND ([SurfaceFormat] != 'RAW')
[0,63]		contains bits [26:21] of the number of entries in the	CHV, BSW	[SurfaceType] == 'SURFTYPE_STRBUF'

RENDER_SURFACE_STATE

		buffer - 1

Programming Notes

The **Depth** of a render target must be the same as the **Depth** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER).

For SURFTYPE_CUBE: For Sampling Engine Surfaces, the range of this field is [0,340], indicating the number of cube array elements (equal to the number of underlying 2D array elements divided by 6). For other surfaces, this field must be zero.

For SURFTYPE_BUFFER: The range of this field is [0,63] unless the Surface Format is RAW and Surface Ptich is 1 byte.

For SURFTYPE_1D, 2D, and CUBE: The range of this field is reduced by one for each increase from zero of **Minimum Array Element**. For example, if **Minimum Array Element** is set to 1024 on a 2D surface, the range of this field is reduced to [0,1023].

20 **Reserved**

Project:	CHV, BSW
Format:	MBZ

19 **Reserved**

Project:	
Format:	MBZ

18 **Reserved**

Project:	CHV, BSW
Format:	MBZ

17:0 **Surface Pitch**

Format:	U18-1 Pitch in #Bytes

Range

1. For surfaces of type SURFTYPE_BUFFER: [0,2047] -> [1B, 2048B]
2. For surfaces of type SURFTYPE_STRBUF: [0,2047] -> [1B, 2048B]
3. For other linear surfaces: [0, 262143] -> [1B, 256KB]
4. For X-tiled surface: [511, 262143] -> [512B, 256KB] = [1 tile, 512 tiles]
5. For Y-tiled surfaces: [127, 262143]->[128B, 256KB] = [1 tile, 2048 tiles]
6. For W-tiled surfaces: [127, 262143]->[128B, 256KB] = [1 tile, 2048 tiles]
7. For TileYF and TileYS surfaces, the range is dependent on the Cu parameter (refer to *Memory Data Formats* section for the definition of the Cu parameter depending on the case). The range in bytes is [2^{Cu}-1,262143] -> [(2^{Cu})B,256KB] = [1 tile, 256KB/(2^{Cu}) tiles]

This field specifies the surface pitch in (#Bytes - 1).
For surfaces of type SURFTYPE_BUFFER and SURFTYPE_STRBUF, this field indicates the size of the structure.

Programming Notes

RENDER_SURFACE_STATE

- For linear *render target* surfaces and surfaces accessed with the typed data port messages, the pitch must be a multiple of the element size for non-YUV surface formats. Pitch must be a multiple of 2 * element size for YUV surface formats.
- For untyped data port messages, which are only supported with **Surface Type** SURFTYPE_BUFFER, the pitch is ignored and assumed to be 1 byte.
- For linear surfaces with **Surface Type** of SURFTYPE_STRBUF, the pitch must be a multiple of 4 bytes.
- For linear surfaces with **Surface Type** of SURFTYPE_BUFFER and **Surface Format** RAW, the pitch must be 1 byte.
- For other linear surfaces, the pitch can be any multiple of bytes.
- For tiled surfaces, the pitch must be a multiple of the tile width.

If the surface is a stencil buffer (and thus has **Tile Mode** set to TILEMODE_WMAJOR), the pitch must be set to 2x the value computed based on width, as the stencil buffer is stored with two rows interleaved. For details on the separate stencil buffer storage format in memory, see GPU Overview (vol1a), Memory Data Formats, Surface Layout, 2D Surfaces, Stencil Buffer Layout (section 8.20.4.8).

4 — 31 — Reserved

Project:	CHV, BSW
Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	MBZ

30:29 — Render Target And Sample Unorm Rotation

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

For Render Target Surfaces: This field specifies the rotation of this render target surface when being written to memory.

For Other Surfaces: This field is ignored.

Value	Name	Description
0h	0DEG	No rotation (0 degrees)
1h	90DEG	Rotate by 90 degrees
3h	270DEG	Rotate by 270 degrees

Programming Notes

Programming Notes for Render Target Surfaces only

- Rotation is not supported for render targets of any type other than simple, non-mip-mapped, non-array 2D surfaces. The surface must be using tiled with X major.
- **Width** and **Height** fields apply to the dimensions of the surface before rotation.
- For 90 and 270 degree rotated surfaces, the **Height** (rather than the **Width**) must be less than or equal to the **Surface Pitch** (specified in bytes).

RENDER_SURFACE_STATE

- For 90 and 270 degree rotated surfaces, the actual **Height** and **Width** of the surface in pixels (not the field value which is decremented) must both be even.

Rotation is supported only for surfaces with the following surface formats: B5G6R5_UNORM, B5G6R5_UNORM_SRGB, R8G8B8A8_UNORM, R8G8B8A8_UNORM_SRGB, B8G8R8[A|X]8_UNORM, B8G8R8[A|X]8_UNORM_SRGB, B10G10R10[A|X]2_UNORM, B10G10R10A2_UNORM_SRGB, R10G10B10A2_UNORM, R10G10B10A2_UNORM_SRGB, R16G16B16A16_FLOAT, R16G16B16X16_FLOAT

28:18 **Minimum Array Element**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	U11

17:7 **Render Target View Extent**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'
Format:	U11-1

Range [0,2047] to indicate extent of [1,2048]
For Render Target and Typed Dataport 3D Surfaces:
This field indicates the extent of the accessible 'R' coordinates minus 1 on the LOD currently being rendered to.
For Render Target and Typed Dataport 1D and 2D Surfaces:
This field must be set to the same value as the Depth field.
For Other Surfaces:
This field is ignored.

6 **Multisampled Surface Storage Format**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

This field indicates the storage format of the multisampled surface.

Value	Name	Description	Project
0h	MSS	Multsampled surface was/is rendered as a render target	All
1h	DEPTH_STENCIL	Multisampled surface was rendered as a depth or stencil buffer	All

Programming Notes

- All multisampled render target surfaces must have this field set to MSFMT_MSS
- IF this field is MSFMT_DEPTH_STENCIL, the only sampling engine messages allowed are "ld2dms", "resinfo", and "sampleinfo".
- This field is ignored if **Number of Multisamples** is MULTISAMPLECOUNT_1

5:3 **Number of Multisamples**

Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

RENDER_SURFACE_STATE

This field indicates the number of multisamples on the surface.

Value	Name	Project
0h	MULTISAMPLECOUNT_1	All
1h	MULTISAMPLECOUNT_2	All
2h	MULTISAMPLECOUNT_4	All
3h	MULTISAMPLECOUNT_8	All
4h	Reserved	CHV, BSW
5h-7h	Reserved	

Programming Notes	Project
If this field is any value other than MULTISAMPLECOUNT_1, the **Surface Type** must be SURFTYPE_2D This field must be set to MULTISAMPLECOUNT_1 unless the surface is a Sampling Engine surface or Render Target surface.	
If this field is any value other than MULTISAMPLECOUNT_1, Surface Min LOD, Mip Count / LOD, and Resource Min LOD must be set to zero.	CHV, BSW

31:0 **Reserved**

Exists If:	[Surface Type] == 'SURFTYPE_STRBUF'
Format:	MBZ

2:0 **Multisample Position Palette Index**

Project:	CHV, BSW
Exists If:	[Surface Type] != 'SURFTYPE_STRBUF'

This field indicates the index into the sample position palette that the multisampled surface is using. This field is only used as a return value for the sampleinfo message, and is otherwise not used by hardware.

Value	Name
[0,7]	

5 31:25 **X Offset**

Format:	PixelOffset[8:2]

This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the surface.
This field effectively loosens the alignment restrictions on the origin of tiled surfaces. Previously, tiled surface origin was (by definition) located at the base address, and thus needed to satisfy the 4KB base address alignment restriction. Now the origin can be specified at a finer (4-wide x 4-high pixel) resolution.

Value	Name	Description
[0,508]		In multiples of 4 (low 2 bits missing)

Programming Notes	Project
• For linear surfaces, this field must be zero.	

RENDER_SURFACE_STATE

- For surfaces accessed with the *Data Port Media Block Read/Write* message, the pixel size is assumed to be 32 bits in width.
- For surfaces accessed with the **Data Port Transpose Read message**, the pixel size is assumed to be 32 bits in width.
- For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero.
- If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero.
- If **Surface Type** not SURFTYPE_2D, this field must be zero.
- If **MIP Count** is not zero, this field must be zero.
- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be zero.
- If **Surface Array** is enabled, this field must be zero.
- If **Auxiliary Surface Mode** is not AUX_NONE, this field must be zero.
- If **Surface Vertical Alignment** is VALIGN_8, this field must be a multiple of 8.
- For **Surface Format** with 8 bits per element, this field must be a multiple of 16.
- For **Surface Format** with 16 bits per element, this field must be a multiple of 8.

This field must be zero if Surface Format is PLANAR*.	CHV, BSW
If sampling an ASTC surface with block size of 5X5 or 5X4 and 0 < [(max(Suface_Width»1,1)%10] < 6, and accessing LOD=2 or higher, then this field must be programmed to 4 pixels.	CHV, BSW

24 **Reserved**

Format:	MBZ

23:21 **Y Offset**

Format:	RowOffset[4:2]

This field specifies the vertical offset in rows from the **Surface Base Address** to the start of the surface. (See additional description in the **X Offset** field.)

Value	Name	Description
[0,28]		In multiples of 4 (low two bits missing)

Programming Notes	Project
• For linear surfaces, this field must be zero. • For render targets in which the **Render Target Array Index** is not zero, this field must be zero. • For **Surface Format** with other than 8, 16, 32, 64, or 128 bits per pixel, this field must be zero. • If **Render Target Rotation** is set to other than RTROTATE_0DEG, this field must be zero.	

RENDER_SURFACE_STATE

- If **Surface Type** not SURFTYPE_2D, this field must be zero.
- If **MIP Count** is not zero, this field must be zero.
- If **Number of Multisamples** is not MULTISAMPLECOUNT_1, this field must be zero.
- If **Surface Array** is enabled, this field must be zero.
- If **Auxiliary Surface Mode** is not AUX_NONE, this field must be zero.

This field must be zero if **Surface Format** is PLANAR*.	CHV, BSW

20 **EWA Disable For Cube**

Project:	CHV, BSW
Format:	Disable

Specifies if EWA mode for LOD quality improvement needs to be disabled for cube maps.

Value	Name	Description
0h	Enable **[Default]**	EWA is enabled for cube maps
1h	Disable	EWA is disabled for cube maps

Programming Notes

This field indicates if EWA mode for LOD quality improvement needs to be disabled for cube maps. By default EWA would be on for cube maps hence this field must be 0. If there is any spec violation seen with EWA on cube maps then this field must be set to 1 to disable EWA for cubes.

19:18 **Reserved**

Project:	CHV, BSW
Format:	MBZ

17:16 **Reserved**

Project:	CHV, BSW
Format:	MBZ

15 **Reserved**

Format:	MBZ

14 **Coherency Type**

Specifies the type of coherency maintained for this surface.

Value	Name	Description	Project
0h	GPU coherent	Surface memory is kept coherent with GPU threads using GPU read/write ordering rules. Surface memory is backed by system memory but is not kept coherent with CPU (LLC).	All
1h	IA coherent	Surface memory is kept coherent with CPU (LLC).	All

RENDER_SURFACE_STATE

		Programming Notes This field may optionally be 1 (IA coherent) for messages sent to SFID_DP_DC0 or SFID_DP_DC1 or SFID_DP_DC2. This field is typically set to 0 (GPU coherent) if the context is operating in a non-SVM legacy mode (for example, Ring Buffer or a Execlist using 32-bit Virtual Address Legacy Context PPGTT32).
	13:12	**Reserved** Format: MBZ
	11:8	**Reserved** Project: CHV, BSW Format: MBZ
	7:4	**Surface Min LOD** Format: U4 In LOD Units **For Sampling Engine and Typed Surfaces:** This field indicates the most detailed LOD that can be accessed as part of this surface. This field is added to the delivered LOD (*sample_l, ld*, or *resinfo* message types) before it is used to address the surface. **For Other Surfaces:** This field is ignored. **Programming Notes** This field must be zero if the **Surface Format** is MONO8
	3:0	**MIP Count / LOD** Format: **Sampling Engine and Typed Surfaces:** U4 in (LOD units - 1) **Render Target Surfaces:** U4 in LOD units Range: **Sampling Engine and Typed Surfaces:** [0,14] representing [1,15] MIP levels **Render Target Surfaces:** [0,14] representing LOD **Other Surfaces:** [0] **For Sampling Engine and Typed Surfaces:** This field indicates the number of MIP levels allowed to be accessed starting at **Surface Min LOD**, which must be less than or equal to the number of MIP levels actually stored in memory for this surface. For sample* messages, the mip map access is clamped to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here. For ld* messages, out-of-bounds behavior results for LODs outside of the range specified in this field. **For Render Target Surfaces:** This field defines the MIP level that is currently being rendered into. This is the absolute MIP level on the surface and is not relative to the **Surface Min LOD** field, which is ignored for render target surfaces. **For Other Surfaces:** This field is reserved : MBZ **Programming Notes**

RENDER_SURFACE_STATE

		The **LOD** of a render target must be the same as the **LOD** of the other render target(s) and of the depth buffer (defined in 3DSTATE_DEPTH_BUFFER). For render targets with YUV surface formats, the **LOD** must be zero. For sampling engine surfaces with YCRCB* or PLANAR* surface format, **MIP Count** must be zero.
6	31	**Reserved** Exists If: ([Surface Format] != 'PLANAR') Format: MBZ
	31	**Separate UV Plane Enable** Exists If: ([Surface Format] == 'PLANAR') Format: Enable If enabled, this field indicates that the U and V are present as separate planes. If disabled, the UV data is interleaved on a single plane. **Programming Notes** — This field must be disabled (separate UV planes are not supported). — **Project**: CHV, BSW
	30	**Reserved** Project: Exists If: ([Surface Format] == 'PLANAR') Format: MBZ
	30:16	**Auxiliary Surface QPitch** Exists If: ([Surface Format] != 'PLANAR') Format: QPitch[16:2] This field specifies the distance in rows between array slices on the auxiliary surface. **Value**: [4h,1FFFCh] — **Name**: — **Description**: in multiples of 4 (low 2 bits missing) **Programming Notes** — This field must be set to an integer multiple of the **Surface Vertical Alignment** — **Project**: Software must ensure that this field is set to a value sufficiently large such that the array slices in the auxiliary surface do not overlap. Refer to the Memory Data Formats section for information on how surfaces are stored in memory. For non-multisampled render target's auxiliary surface, MCS, QPitch must be computed with Horizontal Alignment = 256 and Surface Vertical Alignment = 128. These alignments are only for MCS buffer and not for associated render target. — **Project**: CHV, BSW
	29:16	**X Offset for U or UV Plane** Exists If: ([Surface Format] == 'PLANAR') Format: U14 This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the U plane or interleaved UV plane, depending on the setting of **Separate UV Plane**

RENDER_SURFACE_STATE

Enable.

Programming Notes
This field must be a multiple of 4 (bits 1:0 MBZ).
Auxiliary Surface Mode is forced to AUX_NONE.

15 **Reserved**

Project:	
Format:	MBZ

14 **Reserved**

Project:	All
Exists If:	([Surface Format] == 'PLANAR')
Format:	MBZ

14:12 **Reserved**

Exists If:	([Surface Format] != 'PLANAR')
Format:	MBZ

11:3 **Auxiliary Surface Pitch**

Project:	CHV, BSW
Exists If:	([Surface Format] != 'PLANAR')
Format:	U9-1 Pitch in #Tiles

This field specifies the Auxiliary surface pitch in (#Tiles - 1).

Value	Name	Description
[0, 511]		-> [1 tile, 512 tiles]

13:0 **Y Offset for U or UV Plane**

Exists If:	([Surface Format] == 'PLANAR')
Format:	U14

This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the U plane or interleaved UV plane, depending on the setting of **Separate UV Plane Enable**.

Programming Notes
Auxiliary Surface Mode is forced to AUX_NONE.

Workaround
For formats PLANAR_420_* when this field is not a multiple of 4 the Out-of-Bounds Supression check must be disabled (HALF_SLICE_CHICKEN3[5] must be set to 1) to avoid false out of bound detection.

2:0 **Auxiliary Surface Mode**

Project:	CHV, BSW
Exists If:	([Surface Format] != 'PLANAR')
Format:	U3

RENDER_SURFACE_STATE

Specifies what type of surface the Auxiliary surface is. The Auxiliary surface has its own base address and pitch, but otherwise shares or overrides other fields set for the primary surface, detailed in the programming notes below.

Value	Name	Description	Project
0h	AUX_NONE	No Auxiliary surface is used	All
1h	AUX_MCS	The Auxiliary surfaces is an MCS (Multisample Control Surface)	CHV, BSW
2h	AUX_APPEND	The Auxiliary surface is an append buffer	All
3h	AUX_HIZ	The Auxiliary surface is a hierarchical depth buffer	All
4h	Reserved		
5h	Reserved		CHV, BSW
6h-7h	Reserved		

Programming Notes	Project
The CCS and hierarchical depth Auxiliary surface shares **Height, Width, Depth, Surface Type, Surface Array, Surface Min LOD, MIP Count / LOD, Surface Object Control State, Resource Min LOD**, and **Minimum Array Element** with the primary surface. The hierarchical depth Auxiliary surface uses **Surface Horizontal Alignment** of 16, **Surface Vertical Alignment** of 8, regardless of the primary surface's values for these fields. **X & Y Offset** are set to zero for the purpose of accessing the Auxiliary surface. If this field is set to AUX_HIZ, **Surface Format** must be be one of the following: R32_FLOAT, R24_UNORM_X8_TYPELESS, or R16_UNORM, and the format must match the format used when the surface was used as a depth buffer (with R channel corresponding to D channel).	
The CCS Auxiliary surface for non-multisampled render targets has Horizontal Alignment = 256 and Vertical alignment = 128.	CHV, BSW
If this field is set to AUX_HIZ, **Number of Multisamples** must be MULTISAMPLECOUNT_1, and Surface Type cannot be SURFTYPE_3D.	

7 | 31

Red Clear Color

Project:	CHV, BSW
Format:	Clear Color [CHV, BSW] Enumerated Type

For Sampling Engine Multisampled Surfaces and Render Targets:

Specifies the clear value for the red channel.

For Other Surfaces:

This field is ignored.

30

Green Clear Color

Project:	CHV, BSW
Format:	Clear Color [CHV, BSW] Enumerated Type

For Sampling Engine Multisampled Surfaces and Render Targets:

RENDER_SURFACE_STATE

Specifies the clear value for the green channel.
For Other Surfaces:
This field is ignored.

29 **Blue Clear Color**

Project:	CHV, BSW
Format:	Clear Color [CHV, BSW] Enumerated Type

For Sampling Engine Multisampled Surfaces and Render Targets:
Specifies the clear value for the blue channel.
For Other Surfaces:
This field is ignored.

28 **Alpha Clear Color**

Project:	CHV, BSW
Format:	Clear Color [CHV, BSW] Enumerated Type

For Sampling Engine Multisampled Surfaces and Render Targets:
Specifies the clear value for the alpha channel.
For Other Surfaces:
This field is ignored.

27:25 **Shader Channel Select Red**

Format:	Shader Channel Select [CHV, BSW] Enumerated Type

Specifies which surface channel is read or written in the Red shader channel.

Programming Notes	Project
The Shader channel selects also define which shader channels are written to which surface channel. If the Shader channel select is SCS_ZERO or SCS_ONE then it is not written to the surface. If the shader channel select is SCS_RED it is written to the surface red channel and so on. If more than one shader channel select is set to the same surface channel only the first shader channel in RGBA order will be written. Each shader channel select must be set to the same surface channel (R = SCS_RED, G = SCS_GREEN, B = SCS_BLUE, A = SCS_ALPHA) if the surface is accessed via the sampler's sample_unorm* or sample_8x8 messages.	
The Shader Channel Select fields do not affect the following sampling engine message types: resinfo, sampleinfo, LOD, and ld_mcs. These messages behave as if each Shader Channel Select is set to the same color surface channel.	
For the sampling engine *gather4** messages, the Gather4 Source Channel Select field in the message header defines which channel's Shader Channel Select is used to select the surface channel to be sampled. Other Shader Channel Select fields are ignored.	
For the sampling engine *sample*_c* and *gather4*_c* messages, the compare operation always occurs on the red channel from the surface regardless of the setting of the Shader Channel Select fields.	
For Render Target, Red, Green and Blue Shader Channel Selects MUST be such that	CHV,

RENDER_SURFACE_STATE

		only valid components can be swapped i.e. only change the order of components in the pixel. Any other values for these Shader Channel Select fields are not valid for Render Targets. This also means that there MUST not be multiple shader channels mapped to the same RT channel.	BSW
		When multiple Channel selects have the same value and shader channel is disabled, disable channel writes 0s to memory. This behavior does not match with Data Port message via HDC.	CHV, BSW
	24:22	**Shader Channel Select Green** Format: Shader Channel Select [CHV, BSW] Enumerated Type See **Shader Channel Select Red** for details.	
	21:19	**Shader Channel Select Blue** Format: Shader Channel Select [CHV, BSW] Enumerated Type See **Shader Channel Select Red** for details.	
	18:16	**Shader Channel Select Alpha** Format: Shader Channel Select [CHV, BSW] Enumerated Type See **Shader Channel Select Red** for details.	

Programming Notes	Project
Shader Channel Select Alpha must be set to SCS_ONE for the following formats when sampling (not reading via data port): BC6H_SF16 BC6H_UF16 R32G32B32_FLOAT R11G11B10_FLOAT L32X32_FLOAT PLANAR_420_8 ETC1_RGB8 ETC2_RGB8 EAC_R11 EAC_RG11 EAC_SIGNED_R11 EAC_SIGNED_RG11 ETC2_SRGB8 R8G8B8_UNORM_SRGB R8G8B8_UNORM R8G8B8_SNORM R8G8B8_UINT R8G8B8_SINT R16G16B16_FLOAT R16G16B16_UNORM R16G16B16_SNORM R16G16B16_UINT	CHV, BSW

RENDER_SURFACE_STATE

		R16G16B16_SINT |
		For Render Target, this field MUST be programmed to value = SCS_ALPHA. | CHV, BSW
	15:12	**Reserved** Format: | MBZ
	11:0	**Resource Min LOD** Format: | U4.8 in LOD units **For Sampling Engine Surfaces:** This field indicates the most detailed LOD that is present in the resource underlying the surface. Refer to the "LOD Computation Pseudocode" section for the use of this field. **For Other Surfaces:** This field is ignored. Value | Name [0,14] |
		Programming Notes This field must be zero if the **Surface Format** is MONO8 This field must be zero if the **ChromaKey Enable** is enabled in the associated sampler.
8..9	63:0	**Surface Base Address** Format: | GraphicsAddress[63:0]SurfaceBase Specifies the byte-aligned base address of the surface.

Programming Notes

- For SURFTYPE_BUFFER render targets, this field specifies the base address of first element of the surface. The surface is interpreted as a simple array of that single element type. The address must be naturally-aligned to the element size (e.g., a buffer containing R32G32B32A32_FLOAT elements must be 16-byte aligned).
- For SURFTYPE_BUFFER non-rendertarget surfaces, this field specifies the base address of the first element of the surface, computed in software by adding the surface base address to the byte offset of the element in the buffer. The base address must be aligned to element size.
- Linear depth buffer surface base addresses must be 64-byte aligned. Note that while render targets (color) can be SURFTYPE_BUFFER, depth buffers cannot.
- Mipmapped surfaces are stored in a "monolithic" (fixed) format, and only require a single address for the base MIP. All other MIPs are positioned relative to the base MIP.
- The Base Address for linear (non-tiled) render target surfaces and surfaces accessed with the typed surface read/write data port messages must be element-size aligned for Non-YUV surface formats, or a multiple of 2 element-sizes for YUV surface formats.

RENDER_SURFACE_STATE

- Other linear (non-tiled) surfaces have no alignment requirements (byte alignment is sufficient).
- For tiled surfaces, the actual start of the surface can be offset from the Surface Base Address by the X Offset and Y Offset fields. Tiles are inherently page-aligned (4K or 64K).
- Certain message types used to access surfaces have more stringent alignment requirements. Please refer to the specific data-port message documentation for additional restrictions.

10..11	63:62	**Reserved** Exists If: ([Surface Format] == 'PLANAR') Format: MBZ
	61:48	**X Offset for V Plane** Exists If: ([Surface Format] == 'PLANAR') Format: U14 This field specifies the horizontal offset in pixels from the **Surface Base Address** to the start (origin) of the V plane. **Programming Notes** This field must be a multiple of 4 (bits 1:0 MBZ). This field is ignored if **Separate UV Plane Enable** is disabled.
	47:46	**Reserved** Exists If: ([Surface Format] == 'PLANAR') Format: MBZ
	45:32	**Y Offset for V Plane** Exists If: ([Surface Format] == 'PLANAR') Format: U14 This field specifies the vertical offset in rows from the **Surface Base Address** to the start (origin) of the V plane. **Programming Notes** This field is ignored if **Separate UV Plane Enable** is disabled.
	31:21	**Auxiliary Table Index for Media Compressed Surface** Exists If: [Memory Compression Enable] ==1 This field is valid only if Media Memory Compression is on for the surface(Memory Compression Enable == 1). In that case, the Auxiliary Surface Base address is never expected to be used and hence can be overloaded. This represents the 11 bit index into the table in memory which maps the surface to the auxiliary base address.
	63:12	**Auxiliary Surface Base Address**

RENDER_SURFACE_STATE

		Exists If: ([Surface Format] != 'PLANAR') AND [Memory Compression Enable] == 0 Format: GraphicsAddress[63:12] Specifies the 4kbyte-aligned base address of the Auxiliary surface associated with the primary surface specified in other SURFACE_STATE fields.
	11	**Reserved** Format: MBZ
	10	**Reserved** Project: CHV, BSW Format: MBZ
	9:0	**Reserved** Project: CHV, BSW Format: MBZ
12		
13	31:0	**Reserved** Project: CHV, BSW Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ
14	31:0	**Reserved** Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ
15	31:0	**Reserved** Project: CHV, BSW Exists If: [Auxiliary Surface Mode] == 'AUX_HIZ' Format: MBZ

Render Data Port Message Types

MT_DP_RT - Render Data Port Message Types

Project:	CHV, BSW
Source:	Render Cache DataPort
Size (in bits):	5
Default Value:	0x0000000C

Lists all the Message Types in a Render Data Port Message Descriptor [18:14].

DWord	Bit	Description
0	4	**Reserved** Project: All Format: MBZ Ignored
	3:0	**Message Type** Project: All Format: Enumeration Specifies type of message

Value	Name	Description	Project
0Ch	MT_RTW **[Default]**	Render Target Write message	All
0Dh	MT_RTR	Render Target Read message	All
Others	Reserved	Ignored	All

Render Target Index Message Header Control

MHC_RT_RTI - Render Target Index Message Header Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:3	**Reserved** Project: All Format: Ignore Ignored
	2:0	**Render Target Index** Project: All Format: U3 Specifies the render target index that will be used to select blend state from BLEND_STATE.

Render Target Message Header

MH_RT - Render Target Message Header

Project: CHV, BSW

Source: PRM

Size (in bits): 512

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Render Target Controls 0** Project: All Format: MHC_RT_C0 [CHV, BSW] Specifies controls for Render Target Write and Read messages.
0.1	31:0	**Color Calculator State Pointer** Project: All Format: MHC_RT_CCSP [CHV, BSW] For Render Target Write message, specifies the HWORD-aligned GeneralStateOffset for Color State. Ignored by Render Target Read message.
0.2	31:0	**Render Target Index** Project: All Format: MHC_RT_RTI [CHV, BSW] For Render Target Write message, specifies the render target index used to select blend state from BLEND_STATE. Ignored by Render Target Read message.
0.3-0.4	63:0	**Reserved** Project: All Format: Ignore Ignored
0.5	31:0	**Color Code** Project: All Format: MHC_RT_CC [CHV, BSW] Hardware uses to track synchronizing events and free resources on thread completion.
0.6-0.7	63:0	**Reserved** Project: All

MH_RT - Render Target Message Header

		Format: Ignore Ignored
1.0-1.1	63:0	**Reserved** Project: All Format: Ignore Ignored
1.2	31:0	**Subspan 0** Project: All Format: MHC_RT_SUBSPAN [CHV, BSW] Upper left corner of subspan 0
1.3	31:0	**Subspan 1** Project: All Format: MHC_RT_SUBSPAN [CHV, BSW] Upper left corner of subspan 1
1.4	31:0	**Subspan 2** Project: All Format: MHC_RT_SUBSPAN [CHV, BSW] Upper left corner of subspan 2
1.5	31:0	**Subspan 3** Project: All Format: MHC_RT_SUBSPAN [CHV, BSW] Upper left corner of subspan 3
1.6	31:0	**Reserved** Project: All Format: Ignore Ignored
1.7	31:0	**Pixel Sample Enables** Project: All Format: MHC_RT_PSM [CHV, BSW] Pixel Sample Enables

Render Target Message Header Control

MHC_RT_C0 - Render Target Message Header Control

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31	**Reserved** Project: All Format: Ignore Ignored
	30:27	**Viewport Index** Project: All Format: U4 For Render Target Write message, specifies the index of the viewport currently being used. Range = [0,15] Ignored by Render Target Read message.
	26:16	**Render Target Array Index** Project: All Format: U11 Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index. Range = [0,511] SURFTYPE_2D: specifies the array index. Range = [0,511] SURFTYPE_3D: specifies the Z or R coordinate. Range = [0,2047] SURFTYPE_BUFFER: must be zero. SURFTYPE_CUBE: specifies the face identifier. Mapping (0,+x) (1,-x) (2,+y) (3,-y) (4,+z) (5,-z). **Programming Notes** The Render Target Array Index used by hardware for access to the Render Target is overridden with the Minimum Array Element defined in SURFACE_STATE if it is out of the range between Minimum Array Element and Depth. For cube surfaces, a depth value of 5 is used for this determination.
	15	**Front/Back Facing Polygon** Project: All Format: U1 Determines whether the polygon is front or back facing. Used by the render cache to determine which stencil test state to use. Value: 0h — Name: Front facing — Description: All — Project: All Value: 1h — Name: Back facing — Description: All — Project: All

MHC_RT_C0 - Render Target Message Header Control

14 **Stencil Present to Render Target**

Project:	All
Format:	Enable

For Render Target Write message, indicates that computed stencil is included in the message. Must be zero for Render Target Read message.

13 **Source Depth Present to Render Target**

Project:	All
Format:	Enable

For Render Target Write Message, indicates that source depth data is included in the message. Must be zero for Render Target Read message.

12 **oMask to Render Target**

Project:	All
Format:	Enable

For Render Target Write message, indicates that oMask data is present in the message and is to be used to mask off samples. Must be zero for Render Target Read message.

11 **Source0 Alpha Present to Render Target**

Project:	All
Format:	Enable

For Render Target Write message, indicates that Source0 Alpha (aka o0.a) data is included in RTWrite message. If present, these alpha values are used as inputs to AlphaTest and AlphaToCoverage functions. This is required to meet the API rules when writing to multiple render targets (MRTs). Must be zero for Render Target Read message.

Programming Notes

This bit should not be set when write to RT0, though sending and using redundant alpha will provide the correct results (at lower performance). This bit is not supported on Dual-Source Blend message types, as source0 alpha is already included in those messages. This bit is not supported on replicated data message types.

10 **Reserved**

Project:	All
Format:	Ignore

Ignored

9 **Reserved**

Project:	CHV, BSW
Format:	Ignore

Ignored

MHC_RT_C0 - Render Target Message Header Control

	8:6	**Starting Sample Pair Index** Project: CHV, BSW Format: U3 Indicates the index of the first sample pair of the dispatch. Range = [0,3]
	5:0	**Reserved** Project: All Format: Ignore Ignored

Replicated Pixel Render Target Data Payload Register

MDPR_RGBA - Replicated Pixel Render Target Data Payload Register

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Red** Project: All Format: U32 Specifies the value of all slots' red channel.
1	31:0	**Green** Project: All Format: U32 Specifies the value of all slots' green channel.
2	31:0	**Blue** Project: All Format: U32 Specifies the value of all slots' blue channel.
3	31:0	**Alpha** Project: All Format: U32 Specifies the value of all slots' alpha channel.
4-7	127:0	**Reserved** Project: All Format: Ignore Ignored

Replicated SIMD16 Render Target Data Payload

MDP_RTW_16REP - Replicated SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**RGBA** Project: All Format: MDPR_RGBA [CHV, BSW] RGBA for all slots [15:0]

Reversed SIMD Mode 2 Message Descriptor Control Field

MDC_SM2R - Reversed SIMD Mode 2 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Project: All Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description	Project
00h	SIMD16	SIMD16	All
01h	SIMD8	SIMD8	All

RoundingPrecisionTable_3_Bits

RoundingPrecisionTable_3_Bits

Project: All
Source: PRM
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2:0	**Rounding Precision** Format: U3 Value / Name: 000b +1/16 001b +2/16 010b +3/16 011b +4/16 100b +5/16 101b +6/16 110b +7/16 111b +8/16

S0A SIMD8 Render Target Data Payload

MDP_RTW_A8 - S0A SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

S0A SIMD16 Render Target Data Payload

MDP_RTW_A16 - S0A SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2560
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Source 0 Alpha[15:7]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Source 0 Alpha
2.0-2.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
3.0-3.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red

<table>
<tr><th colspan="3">MDP_RTW_A16 - S0A SIMD16 Render Target Data Payload</th></tr>
<tr><td>4.0-4.7</td><td>255:0</td><td>Green[7:0]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [7:0] Green</td></tr>
<tr><td>5.0-5.7</td><td>255:0</td><td>Green[15:8]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [15:8] Green</td></tr>
<tr><td>6.0-6.7</td><td>255:0</td><td>Blue[7:0]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [7:0] Blue</td></tr>
<tr><td>7.0-7.7</td><td>255:0</td><td>Blue[15:8]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [15:8] Blue</td></tr>
<tr><td>8.0-8.7</td><td>255:0</td><td>Alpha[7:0]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [7:0] Alpha</td></tr>
<tr><td>9.0-9.7</td><td>255:0</td><td>Alpha[15:8]<table><tr><td>Project:</td><td>All</td></tr><tr><td>Format:</td><td>MDP_DW_SIMD8 [CHV, BSW]</td></tr></table>Slots [15:8] Alpha</td></tr>
</table>

SAMPLER_BORDER_COLOR_STATE

SAMPLER_BORDER_COLOR_STATE

Project:	Pre-CHV, BSW
Source:	PRM
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

Description	Project
The interpretation of the border color depends on the Texture Border Color Mode field in SAMPLER_STATE as follows: • DX9 mode: The border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used even for the missing channels. • DX10/OGL mode: the format of the border color depends on the format of the surface being sampled. If the map format is UINT, then the border color format is R32G32B32A32_UINT. If the map format is SINT, then the border color format is R32G32B32A32_SINT. Otherwise, the border color format is R32G32B32A32_FLOAT. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the red channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.The format of this state depends on the Texture Border Color Mode field.	CHV, BSW

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8

SAMPLER_BORDER_COLOR_STATE

		Texture Border Color Mode = DX9
	23:16	**Border Color Blue** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	15:8	**Border Color Green** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	31:0	**Border Color Red - (DX10/0GL)** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/0GL' Format: IEEE_FP Texture Border Color Mode = DX10/OGL
	7:0	**Border Color Red - (DX9)** Exists If: Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
1	31:0	**Border Color Green** Format: IEEE_FP Texture Border Color Mode = DX10/OGL
2	31:0	**Border Color Blue** Format: IEEE_FP Texture Border Color Mode = DX10/OGL
3	31:0	**Border Color Alpha** Format: IEEE_FP Texture Border Color Mode = DX10/OGL

SAMPLER_INDIRECT_STATE_BORDER_COLOR

SAMPLER_INDIRECT_STATE_BORDER_COLOR

Project:	CHV, BSW
Source:	PRM
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure is a one version of the SAMPLER_INDIRECT_STATE structure, suitable for many needs.
An instance of this structure is pointed to by the **Indirect State Pointer** field in SAMPLER_STATE.
The interpretation of the border color depends on the **Texture Border Color Mode** field in SAMPLER_STATE as follows:

- In **DX9** mode, the border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used *even for the missing channels*.
- In **DX10/OGL** mode, the format of the border color is R32G32B32A32_FLOAT, R32G32B32A32_SINT, or R32G32B32A32_UINT, depending on the surface format chosen. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the *red* channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
	23:16	**Border Color Blue As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8

SAMPLER_INDIRECT_STATE_BORDER_COLOR

	15:8	**Border Color Green As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
	31:0	**Border Color Red As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
	31:0	**Border Color Red As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Red As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	7:0	**Border Color Red As U8** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: U8
1	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Green As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Green As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Green As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
2	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Blue As S31**

SAMPLER_INDIRECT_STATE_BORDER_COLOR

		Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Blue As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Blue As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float
3	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Alpha As S31** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsSigned]=='true' Format: S31
	31:0	**Border Color Alpha As U32** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsUnsigned]=='true' Format: U32
	31:0	**Border Color Alpha As Float** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' AND (Structure[RENDER_SURFACE_STATE][Surface Format]Property[IsFloat]=='true' Format: IEEE_Float

SAMPLER_INDIRECT_STATE

SAMPLER_INDIRECT_STATE

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Note: There are three variations of this structure, defined separately because their payloads have different lengths. Currently only SAMPLER_INDIRECT_STATE_BORDER_COLOR is fully defined.
This structure is pointed to by **Indirect State Pointer** (SAMPLER_STATE).
The interpretation of the border color depends on the **Texture Border Color Mode** field in SAMPLER_STATE as follows:

- In **DX9** mode, the border color is 8-bit UNORM format, regardless of the surface format chosen. For surface formats with one or more channels missing (i.e. R5G6R5_UNORM is missing the alpha channel), the value from the border color, if selected, will be used *even for the missing channels*.
- In **DX10/OGL** mode, the format of the border color is R32G32B32A32_FLOAT, R32G32B32A32_SINT, or R32G32B32A32_UINT, depending on the surface format chosen. For surface formats with one or more channels missing, the value from the border color is not used for the missing channels, resulting in these channels resulting in the overall default value (0 for colors and 1 for alpha) regardless of whether border color is chosen. The surface formats with "L" and "I" have special behavior with respect to the border color. The border color value used for the replicated channels (RGB for "L" formats and RGBA for "I" formats) comes from the *red* channel of border color. In these cases, the green and blue channels, and also alpha for "I", of the border color are ignored.

The format of this state depends on the **Texture Border Color Mode** field.

Programming Notes

- DX9 mode is not supported for surfaces with more than 16 bits in any channel, other than 32-bit float formats which are supported.
- The conditions under which this color is used depend on the **Surface Type** - 1D/2D/3D surfaces use the border color when the coordinates extend beyond the surface extent; cube surfaces use the border color for "empty" (disabled) faces.
- The border color itself is accessed through the texture cache hierarchy rather than the state cache hierarchy. Thus, if the border color is changed in memory, the texture cache must be invalidated and the state cache does not need to be invalidated.
- MAPFILTER_MONO: The border color is ignored. Border color is fixed at a value of 0 by hardware.

DWord	Bit	Description
0	31:24	**Border Color Alpha** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8

SAMPLER_INDIRECT_STATE

		Texture Border Color Mode = DX9
	23:16	**Border Color Blue** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	15:8	**Border Color Green** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
	31:0	**Border Color Red** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: SINT32 (2's complement) for all SINT surface formats Format: UINT32 for all UINT surface formats Format: IEEE_FP for all other surface formats Texture Border Color Mode = DX10/OGL
	7:0	**Border Color Red** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: UNORM8 Texture Border Color Mode = DX9
1	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Green** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
2	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Blue**

SAMPLER_INDIRECT_STATE

		Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
3	31:0	**Reserved** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX9' Format: MBZ
	31:0	**Border Color Alpha** Exists If: //Structure[SAMPLER_STATE][Texture Border Color Mode] == 'DX10/OGL' Format: IEEE_FP Format: S31 Format: U32 Texture Border Color Mode = DX10/OGL
4..15	31:0	**Reserved**

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

Project: CHV, BSW

Source: PRM

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description	Project
ExistsIf = AVS	CHV, BSW

DWord	Bit	Description
0	31:24	**Table 0Y Filter Coefficient[n,1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Table 0X Filter Coefficient[n,1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Table 0Y Filter Coefficient[n,0]** Format: S1.6 2's Complement **Range:** [-2, +2) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
	7:0	**Table 0X Filter Coefficient[n,0]** Format: S1.6 2's Complement **Range:** [-2, +2) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
1	31:24	**Table 0Y Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 0X Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	15:8	**Table 0Y Filter Coefficient[n,2]**

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

		Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	7:0	**Table 0X Filter Coefficient[n,2]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
2	31:24	**Table 0Y Filter Coefficient[n,5]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 0X Filter Coefficient[n,5]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	15:8	**Table 0Y Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
	7:0	**Table 0X Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0) **Programming Notes** If the format is R10G10B10A2_UNORM or R8G8B8A8_UNORM, this field MBZ.
3	31:24	**Table 0Y Filter Coefficient[n,7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Table 0X Filter Coefficient[n,7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Table 0Y Filter Coefficient[n,6]** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**Table 0X Filter Coefficient[n,6]** Format: S1.6 2's Complement

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

		Range: [-2, +2)
4	31:24	**Table 1X Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 1X Filter Coefficient[n,2]** Format: S1.6 2's Complement **Range:** [-1.0, +1.0) — CHV, BSW
	15:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:8	**Table 1X Filter Coefficient[n,5]** Format: S1.6 2's Complement **Range:** [-1.0, +1.0) — CHV, BSW
	7:0	**Table 1X Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
6	31:24	**Table 1Y Filter Coefficient[n,3]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**Table 1Y Filter Coefficient[n,2]** Format: S1.6 2's Complement **Range:** [-1.0, +1.0) — CHV, BSW
	15:0	**Reserved** Format: MBZ
7	31:16	**Reserved** Format: MBZ
	15:8	**Table 1Y Filter Coefficient[n,5]**

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

		Format: S1.6 2's Complement **Description** / **Project** **Range:** [-1.0, +1.0) / CHV, BSW
	7:0	**Table 1Y Filter Coefficient[n,4]** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SAMPLER_STATE_8x8_AVS

SAMPLER_STATE_8x8_AVS

Project:	CHV, BSW
Source:	PRM
Size (in bits):	4928
Default Value:	0x0294806C, 0x00000000, 0x39CFD1FF, 0x839F0000, 0x9A6E4000, 0x02601180, 0xFFFE2F2E, 0x00000000, 0xD82E0000, 0x8285ECEC, 0x00008282, 0x00000000, 0x02117000, 0xA38FEC96, 0x00008CC8, 0x00000000

Description	Project
ExistsIf = AVS	CHV, BSW

DWord	Bit	Description
0	31:28	**Reserved** Format: MBZ
	27:23	**Reserved**
	22:18	**Reserved**
	17:12	**Strong Edge Threshold** Default Value: 8 Format: U6

SAMPLER_STATE_8x8_AVS

		If EM > **Strong Edge Threshold**, the basic VSA detects a strong edge.
	11:6	**Weak Edge Threshold** Default Value: 1 Format: U6 If **Strong Edge Threshold** > EM > **Weak Edge Threshold**, the basic VSA detects a weak edge.
	5:0	**Gain Factor** Default Value: 44 Format: U6 User control sharpening strength
1	31:0	**Reserved** Project: CHV, BSW Format: MBZ
2	31:27	Reserved
	26:22	Reserved
	21:17	Reserved
	16:14	**Strong Edge Weight** Default Value: 7 Format: U3 Sharpening strength when a strong edge is found in basic VSA.
	13:11	**Regular Weight** Default Value: 2 Format: U3 Sharpening strength when a weak edge is found in basic VSA.
	10:8	**Non Edge Weight** Default Value: 1 Format: U3 Sharpening strength when no edge is found in basic VSA.
	7:0	**Global Noise Estimation** Default Value: 255 Format: U8 Global noise estimation of previous frame.
3	31	**Reserved**

SAMPLER_STATE_8x8_AVS

	Default Value:	1
	Format:	U1

30 Reserved

Format:	U1

29:28 Enable 8-tap filter

Adaptive Filtering (Mode = 11) ExistsIf:
R10G10B10A2_UNORM R8G8B8A8_UNORM (AYUV also) R8B8G8A8_UNORM B8G8R8A8_UNORM R16G16B16A16

Enable 8-tap Filtering on UV channel (Mode = 10) ExistsIf:
R10G10B10A2_UNORM R8G8B8A8_UNORM (AYUV also) R8B8_UNORM (CrCb) R8_UNORM R8B8G8A8_UNORM B8G8R8A8_UNORM R16G16B16A16 Y8_UNORM

Value	Name	Description
00b		4-tap filter is only done on all channels.
01b		Enable 8-tap Adaptive filter on G-channel. 4-tap filter on other channels.
10b		8-tap filter is done on all channels (UV-ch uses the Y-coefficients)
11b		Enable 8-tap Adaptive filter all channels (UV-ch uses the Y-coefficients).

Programming Notes
For 00 and 10, are applicable for RGB surfaces only or surface without Y-ch. In case it is a YUV surface it will default to adaptive mode automatically which is 01 and 11 respectively. Alpha channel is always bi-linear filter irrespective of the above modes.
Mode 01 and 00 are legacy support and are supported on all surface formats.
When Mode is 10 and Surface format is Y8_UNORM, Bypass X/Y Adaptive Filtering must be 1, and Default Sharp Level must be 255

27:22 Hue_Max

Default Value:	14
Format:	U6

Rectangle half width.

21:16 Sat_Max

Default Value:	31
Format:	U6

Rectangle half length

15:8 Cos(alpha)

SAMPLER_STATE_8x8_AVS

		Format: S0.7 2's Complement Deafult Value: 79/128
	7:0	**Sin(alpha)** Format: S0.7 2's Complement Deafult Value: 101/128
4	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate.
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate.
	15	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	14:12	**Diamond Margin** Default Value: 4 Format: U3
	11	**Reserved** Project: CHV, BSW Format: MBZ
	10:0	**S3U** Format: S2.8 2's Complement Deafult Value: 0/256
5	31	Reserved
	30:24	**Diamond_du** Default Value: 2 Format: S6 2's Complement Rhombus center shift in the sat-direction, relative to the rectangle center.
	23:21	**HS_margin** Default Value: 3

SAMPLER_STATE_8x8_AVS

		Format: U3 Defines rectangle margin
	20:13	**Diamond_alpha** Format: U2.6 Deafault Value: 100/64 1 / tan(β)
	12:7	**Diamond_Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.
	6:0	**Diamond_dv** Default Value: 0 Format: S6 2's Complement Rhombus center shift in the hue-direction, relative to the rectangle center.
6	31:24	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function.
	23:16	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	15:8	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	7:0	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
7	31:16	**Reserved**

SAMPLER_STATE_8x8_AVS

		Format: MBZ
	15:0	**INV_Margin_VYL** Format: U0.16 1/Margin_VYL = 3300/65536
8	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**INV_Margin_VYU** 1/Margin_VYU = 1600/65536
9	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y Point 2 of the lower part of the detection PWLF.
10	31:27	**Y_Slope_2** Format: U2.3

SAMPLER_STATE_8x8_AVS

		Deafault Value: 31/8 Slope between points Y3 and Y4.
	26:16	**S0L** Format: S2.8 2's Complement Deafault Value: -5/256 Slope 0 of the lower part of the detection PWLF.
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8
11	31:22	**Reserved** Format: MBZ
	21:11	**S2L** Format: S2.8 2's Complement Default Value: 0/256 Slope 2 of the lower part of the detection PWLF.
	10:0	**S1L** Format: S2.8 2's Complement Default Value: 0/256 Slope 1 of the lower part of the detection PWLF.
12	31:27	**Y_Slope1** Format: U2.3 Default Value: 31/8 Slope between points Y1 and Y2.
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.

SAMPLER_STATE_8x8_AVS

	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Format: S2.8 2's Complement Default Value: 0/256 Slope 3 of the lower part of the detection PWLF.
13	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8 V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
14	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Format: S2.8 2's Complement Default Value: 256/256 Slope 0 of the upper part of the detection PWLF.
	15:8	**B3U**

SAMPLER_STATE_8x8_AVS

		Default Value: 140 Format: U8 V Bias 3 of the upper part of the detection PWLF.
	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
15	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Format: S2.8 2's Complement Deafult Value: -179/256 Slope 2 of the upper part of the detection PWLF.
	10:0	**S1U** Format: S2.8 2's Complement Deafult Value: 113/256 Slope 1 of the upper part of the detection PWLF.
16..23	255:0	**Filter Coefficient[0]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
24..31	255:0	**Filter Coefficient[1]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
32..39	255:0	**Filter Coefficient[2]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
40..47	255:0	**Filter Coefficient[3]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
48..55	255:0	**Filter Coefficient[4]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
56..63	255:0	**Filter Coefficient[5]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
64..71	255:0	**Filter Coefficient[6]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
72..79	255:0	**Filter Coefficient[7]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]

SAMPLER_STATE_8x8_AVS

80..87	255:0	**Filter Coefficient[8]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
88..95	255:0	**Filter Coefficient[9]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
96..103	255:0	**Filter Coefficient[10]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
104..111	255:0	**Filter Coefficient[11]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
112..119	255:0	**Filter Coefficient[12]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
120..127	255:0	**Filter Coefficient[13]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
128..135	255:0	**Filter Coefficient[14]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
136..143	255:0	**Filter Coefficient[15]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
144..151	255:0	**Filter Coefficient[16]** Format: SAMPLER_STATE_8x8_AVS_COEFFICIENTS [CHV, BSW]
152	31:24	**Default Sharpness Level** Format: U8 When adaptive scaling is off, determines the balance between sharp and smooth scalers. **Value** / **Name** / **Description** 0 / **[Default]** / Contribute 1 from the smooth scalar 255 / / Contribute 1 from the sharp scalar
	23:16	**Max Derivative 4 Pixels** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 4 pixel area.
	15:8	**Max Derivative 8 Pixels** Format: U8 Used in adaptive filtering to specify the lower boundary of the smooth 8 pixel area.
	7	**Reserved** Format: MBZ
	6:4	**Transition Area with 4 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 4 pixel calculation.

SAMPLER_STATE_8x8_AVS

	3	**Reserved** Format: MBZ
	2:0	**Transition Area with 8 Pixels** Format: U3 Used in adaptive filtering to specify the width of the transition area for the 8 pixel calculation.
153	31:23	**Reserved** Format: MBZ
	22	**Bypass X Adaptive Filtering** Format: Disable When disabled, the X direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value. **Value** / **Name** / **Description** 1 / Disble / Disable X Ddaptive Filtering 0 / Enable / Enable X Adaptive Filtering
	21	**Bypass Y Adaptive Filtering** Format: Disable When disabled, the Y direction will use **Default Sharpness Level** to blend between the smooth and sharp filters rather than the calculated value. **Value** / **Name** / **Description** 1 / Disble / Disable Y Ddaptive Filtering 0 / Enable / Enable Y Adaptive Filtering
	20:2	**Reserved** Format: MBZ
	1	**Adaptive Filter for all channels** Format: Enable Only to be enabled if 8-tap Adaptive filter mode is on, eElse it should be disabled. **Value** / **Name** / **Description** 1 / Enable / Enable Adaptive Filter on UV/RB Channels 0 / Disble / Disable Adaptive Filter on UV/RB Channels
	0	**RGB Adaptive** Format: Enable This should be always set to 0 for YUV input and can be enabled/disabled for RGB input. This should be enabled only if we enable 8-tap adaptive filter for RGB input. **Value** / **Name** / **Description** 1 / Enable / Enable the RGB Adaptive filter using the equation (Y=(R+2G+B)»2) 0 / Disble / Disable the RGB Adaptive equation and use G-Ch directly for adaptive filter

SAMPLER_STATE_8x8_CONVOLVE

SAMPLER_STATE_8x8_CONVOLVE	
Project:	CHV, BSW
Source:	PRM
Size (in bits):	16384
Default Value:	0x00000000, 0x00000000,

SAMPLER_STATE_8x8_CONVOLVE
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

SAMPLER_STATE_8x8_CONVOLVE

0x00000000, 0x00000000

Description	Project
Function: 0001b ExistsIf: [Convolve] && [(Kernel Size) =< (15x15)]	CHV, BSW

DWord	Bit	Description
0	31:21	**Reserved** Format: MBZ
	20	**Reserved** Project: CHV, BSW Format: MBZ
	19:17	**Reserved** Format: MBZ
	16	**Reserved** Project: CHV, BSW Format: MBZ
	15:13	**Reserved** Format: MBZ
	12	**Size of the Coefficient** Value 0 (8bit): The lower 8 bits of the accumulator is forced to zero or ignored during the accumulation operation. Value 1 (16bit): The lower 8 bits are also included for the operation. The final result of the accumulator is shifted before clamping the result as specified by the Scale down value.: Result[15:0] = Clamp(Accum[40:12] » scale_down)
	11:8	**Scale down value** Exists If: //[Convolve] Only Value [0,10]: The final result is shifted by this value before clamp is done.
	7:4	**WIDTH** Exists If: //[Convolve] Only It contains the WIDTH of the kernel. Value [2-15]
	3:0	**HEIGHT** Exists If: //[Convolve] Only It contains the HEIGHT of the kernel.

SAMPLER_STATE_8x8_CONVOLVE

		[2-15]
1..15	31:0	**Reserved** Format: MBZ
16	31:16	**Filter Coefficient[0,1]** Exists If: //[Filtering] Operation Format: S3.4(8bit)/S3.12(16bit) in 2's Complement **Range:** [-8.0, +8.0) **Programming Notes** Please note that this field is MBZ if not used in the Filtering Mode.
	15:0	**Filter Coefficient[0,0]** Exists If: //[Filtering] Operation Format: S3.4(8bit)/S3.12(16bit) in 2's Complement **Range:** [-8.0, +8.0) **Programming Notes** Please note that this field is MBZ if not used in the Filtering Mode.
17	31:16	**Filter Coefficient[0,3]** Exists If: //[Filtering] Operation Format: S3.4(8bit)/S3.12(16bit) in 2's Complement **Range:** [-8.0, +8.0) **Programming Notes** Please note that this field is MBZ if not used in the Filtering Mode.
	15:0	**Filter Coefficient[0,2]** Exists If: //[Filtering] Operation Format: S3.4(8bit)/S3.12(16bit) in 2's Complement **Range:** [-8.0, +8.0) **Programming Notes** Please note that this field is MBZ if not used in the Filtering Mode.
18..19	31:0	**Filter Coefficient[0,7:4]** This table has the same layout as shown above.
20..23	31:0	**Filter Coefficient[0,15:8]** This table has the same layout as shown above.
24..143	31:0	**Filter Coefficient[15:1,15:0]** Columns [15:1] of the coefficient containing 16 coefficients for [15:0] rows. This table has the same layout as shown above.
144..263	31:0	**Reserved** Project: CHV, BSW Format: MBZ

SAMPLER_STATE_8x8_CONVOLVE		
264..391	31:0	**Reserved** Project: CHV, BSW Format: MBZ
392..511	31:0	**Reserved** Project: CHV, BSW Format: MBZ

SAMPLER_STATE_8x8_ERODE_DILATE_MINMAXFILTER

SAMPLER_STATE_8x8_ERODE_DILATE_MINMAXFILTER

Project: CHV, BSW

Source: PRM

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description	Project
The table is valid for the following funstions: 0100 - Erode 0101 - Dilate 0011 - MinMaxFilter	CHV, BSW

Programming Notes
Max kernel size is 15x15. For sizes less than 15x15 the coefficients not used should be zeroed out.

DWord	Bit	Description
0	31:16	**16bit Mask for Row0 [15:0]**
	15:8	**Reserved** Format: MBZ
	7:4	**Width Of The Kernel** Value: 2-15; Name:
	3:0	**Height Of The Kernel** Value: 2-15; Name:
1	31:16	**16bit Mask for Row2 [15:0]**
	15:0	**16bit Mask for Row1 [15:0]**
2	31:16	**16bit Mask for Row4 [15:0]**
	15:0	**16bit Mask for Row3 [15:0]**
3	31:16	**16bit Mask for Row6 [15:0]**
	15:0	**16bit Mask for Row5 [15:0]**
4	31:16	**16bit Mask for Row8 [15:0]**
	15:0	**16bit Mask for Row7 [15:0]**
5	31:16	**16bit Mask for Row10 [15:0]**
	15:0	**16bit Mask for Row9 [15:0]**
6	31:16	**16bit Mask for Row12 [15:0]**
	15:0	**16bit Mask for Row11 [15:0]**
7	31:16	**16bit Mask for Row14 [15:0]**
	15:0	**16bit Mask for Row13 [15:0]**

SAMPLER_STATE

SAMPLER_STATE

Project:	CHV, BSW
Source:	PRM
Exists If:	//(MessageType != 'Deinterlace') && (MessageType != 'Sample_8x8')
Size (in bits):	128
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000

This is the normal sampler state used by all messages that use SAMPLER_STATE except sample_8x8 and deinterlace. The sampler state is stored as an array of up to 16 elements, each of which contains the dwords described here. The start of each element is spaced 4 dwords apart. The first element of the sampler state array is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31	**Sampler Disable** Project: All Format: Disable This field allows the sampler to be disabled. If disabled, all output channels will return 0.
	30	**Reserved** Project: CHV, BSW
	29	**Texture Border Color Mode** For some surface formats, the 32 bit border color is decoded differently based on the border color mode. In addition, the default value of channels not included in the surface may be affected by this field. Refer to the "Sampler Output Channel Mapping" table for the values of these channels, and for surface formats that may only support one of these modes. Also refer to the definition of SAMPLER_BORDER_COLOR_STATE for more details on the behavior of the two modes defined by this field.

Value	Name	Description
0h	DX10/OGL	DX10/OGL mode for interpreting the border color
1h	DX9	DX9 and earlier mode for interpreting the border color

Programming Notes	Project
This field is required to be the same for every message over a period of time. A flush of the sampler cache must occur before a message with the opposite state of this field is delivered.	
This field must be set to DX9 mode when used with surfaces that have Surface Format P4A4_UNORM or A4P4_UNORM.	
This field must be set to DX10/OGL mode when used with surfaces that have Surface Format YCRCB_SWAPUV or YCRCB_SWAPY.	
This field must be set to DX10/OGL mode if **Surface Format** for the associated	CHV,

SAMPLER_STATE

	surface is UINT OR SINT.	BSW
	This field must be set to DX10/OGL mode if REDUCTION_MINIMUM or REDUCTION_MAXIMUM or message type is sample_min or sample_max.	

28:27 **LOD PreClamp Mode**

Project:	CHV, BSW

This field determines whether the computed LOD is clamped to [max,min] mip level before the mag-vs-min determination is performed.
PRECLAMP_OGL: LOD pre-clamped to **Min LOD** and **Max LOD**
OpenGL API currently clamps LOD to the **Min LOD** and **Max LOD** (from Sampler State) prior to performing min/mag determination, and therefore it is expected that an OpenGL driver would need to set this field to PRECLAMP_OGL.

Value	Name	Description
0h	NONE	LOD PreClamp disabled
1h	Reserved	
2h	OGL	LOD PreClamp enabled (OGL mode)

26:22 **Base Mip Level**

Project:	CHV, BSW
Format:	U4.1

Range: [0.0, 14.0]
Specifies which mip level is considered the "base" level when determining mag-vs-min filter and selecting the "base" mip level.

21:20 **Mip Mode Filter**

Project:	All
Format:	U2 Enumerated Type

This field determines if and how mip map levels are chosen and/or combined when texture filtering.

Value	Name	Description	Project
0h	NONE	Disable mip mapping - force use of the mipmap level corresponding to Min LOD.	All
1h	NEAREST	Nearest, Select the nearest mip map	All
2h	Reserved		All
3h	LINEAR	Linearly interpolate between nearest mip maps (combined with linear min/mag filters this is analogous to "Trilinear" filtering).	All

Programming Notes	Project
MIPFILTER_LINEAR is not supported for surface formats that do not support	

SAMPLER_STATE

"Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type or minimum/maximum operation.	
Mip Mode Filter must be set to MIPFILTER_NONE or MIPFILTER_NEAREST if Surface Format for the associated surface is UINT or SINT. However, all settings of this field are allowed with UINT/SINT if a minimum or maximum operation is being performed.	CHV, BSW

19:17 **Mag Mode Filter**

Project:	All
Format:	U3 Enumerated Type

This field determines how texels are sampled/filtered when a texture is being "magnified" (enlarged). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension.

Value	Name	Description	Project
0h	NEAREST	Sample the nearest texel	All
1h	LINEAR	Bilinearly filter the 4 nearest texels	All
2h	ANISOTROPIC	Perform an "anisotropic" filter on the chosen mip level	All
4h-5h	Reserved		All
6h	MONO	Perform a monochrome convolution filter	All
7h	Reserved		All

Programming Notes	Project
Only MAPFILTER_NEAREST and MAPFILTER_LINEAR are supported for surfaces of type SURFTYPE_3D.	
Only MAPFILTER_NEAREST is supported for surface formats that do not support "Sampling Engine Filtering" as indicated in the Surface Formats table unless using the sample_c message type or minimum/maximum operation.	
MAPFILTER_MONO: Only CLAMP_BORDER texture addressing mode is supported. . Both Mag Mode Filter and Min Mode Filter must be programmed to MAPFILTER_MONO. Mip Mode Filter must be MIPFILTER_NONE. Only valid on surfaces with Surface Format MONO8 and with Surface Type SURFTYPE_2D.	
MAPFILTER_ANISOTROPIC may cause artifacts at cube edges if enabled for cube maps with the TEXCOORDMODE_CUBE addressing mode.	
MAPFILTER_ANISOTROPIC will be overridden to MAPFILTER_LINEAR when using a sample_l or sample_l_c message type or when Force LOD to Zero is set in the message header.	
Both Mag Mode Filter and Min Mode Filter must be set to MAPFILTER_NEAREST if Surface Format for the associated surface is UINT or SINT. However, all settings of this field other than MAPFILTER_MONO are allowed with UINT/SINT if a minimum or maximum operation is being performed.	CHV, BSW
MAPFILTER_FLEXIBLE might have data corruption when sampled from surface with float32 format with exponent value exceeded 248	CHV, BSW

SAMPLER_STATE

MAPFILTER_FLEXIBLE operates on float16 or float32 surfaces could have erroneous signed for infinity output i.e. 0x7f800000 <-> 0xff800000	CHV, BSW
MAPFILTER_FLEXIBLE when float16 +/-inf apply to coefficient that are absolutely larger than 1.0 output result could be nan instead of +/-inf MAPFILTER_FLEXIBLE: A Null Tile reference will be reported back even if the associated texel has a coefficient of 0.0.	CHV, BSW

16:14 **Min Mode Filter**

Project:	All
Format:	U3 Enumerated Type

This field determines how texels are sampled/filtered when a texture is being "minified" (shrunk). For volume maps, this filter mode selection also applies to the 3rd (inter-layer) dimension.See Mag Mode Filter

Value	Name	Description	Project
0h	NEAREST	Sample the nearest texel	All
1h	LINEAR	Bilinearly filter the 4 nearest texels	All
2h	ANISOTROPIC	Perform an "anisotropic" filter on the chosen mip level	All
4h-5h	Reserved		All
6h	MONO	Perform a monochrome convolution filter	All
7h	Reserved		All

Programming Notes	Project
FLEXIBLE: A Null Tile reference will be reported back even if the associated texel has a coefficient of 0.0.	CHV, BSW

13:1 **Texture LOD Bias**

Project:	All
Format:	S4.8 2's complement

Range: [-16.0, 16.0)
This field specifies the signed bias value added to the calculated texture map LOD prior to min-vs-mag determination and mip-level clamping. Assuming mipmapping is enabled, a positive LOD bias will result in a somewhat blurrier image (using less-detailed mip levels) and possibly higher performance, while a negative bias will result in a somewhat crisper image (using more-detailed mip levels) and may lower performance.

Programming Notes
There is no requirement or need to offset the LOD Bias in order to produce a correct LOD for texture filtering (as was required for correct bilinear and anisotropic filtering in some legacy devices).

0 **Anisotropic Algorithm**

Project:	All

SAMPLER_STATE

		Format: U1 Enumerated Type Controls which algorithm is used for anisotropic filtering. Generally, the EWA approximation algorithm results in higher image quality than the legacy algorithm.

Value	Name	Description	Project
0h	LEGACY	Use the legacy algorithm for anisotropic filtering	All
1h	EWA Approximation	Use the new EWA approximation algorithm for anisotropic filtering	All

1	31:20	**Min LOD** Project: All Format: U4.8 in LOD units Range: [0.0, 14.0], where the upper limit is also bounded by the Max LOD. This field specifies the minimum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and before this maximum (resolution) mip clamping is applied.The integer bits of this field are used to control the "maximum" (highest resolution) mipmap level that may be accessed (where LOD 0 is the highest resolution map). The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use.

Programming Notes

If Min LOD is greater than Max LOD, Min LOD takes precedence, i.e. the resulting LOD will always be Min LOD.

This field must be zero if the Min or Mag Mode Filter is set to MAPFILTER_MONO

	19:8	**Max LOD** Project: All Format: U4.8 in LOD units Range: [0.0, 14.0] This field specifies the maximum value used to clamp the computed LOD after LOD bias is applied. Note that the minification-vs.-magnification status is determined after LOD bias and before this minimum (resolution) mip clamping is applied.The integer bits of this field are used to control the "minimum" (lowest resolution) mipmap level that may be accessed.The fractional bits of this value effectively clamp the inter-level trilinear blend factor when trilinear filtering is in use.Force the mip map access to be between the mipmap specified by the integer bits of the Min LOD and the ceiling of the value specified here.
	7	**ChromaKey Enable** Project: CHV, BSW Format: Enable This field enables the chroma key function.

Programming Notes

SAMPLER_STATE

Supported only on a specific subset of surface formats. See section titled: "Surface Formats" in thsi volume for supported formats. This field must be disabled if min or mag filter is MAPFILTER_MONO or MAPFILTER_ANISOTROPIC. This field must be disabled if used with a surface of type SURFTYPE_3D.

6:5 **ChromaKey Index**

Project:	CHV, BSW
Format:	U2

Range: [0, 3]

This field specifies the index of the ChromaKey Table entry associated with this Sampler. This field is a "don't care" unless **ChromaKey Enable** is ENABLED.

4 **ChromaKey Mode**

Project:	CHV, BSW
Format:	U1 Enumerated Type

This field specifies the behavior of the device in the event of a ChromaKey match. This field is ignored if ChromaKey is disabled.

KEYFILTER_REPLACE_BLACK: In this mode, each texel that matches the chroma key is replaced with (0,0,0,0) (black with alpha=0) prior to filtering. This will tend to darken/fade edges of keyed regions. Note that the pixel pipeline must be programmed to use the resulting filtered texel value to gain the intended effect, e.g., handle the case of a totally keyed-out region (filtered texel alpha==0) through use of alpha test, etc.

Value	Name	Description
0h	KEYFILTER_KILL_ON_ANY_MATCH	In this mode, if any contributing texel matches the chroma key, the corresponding pixel mask bit for that pixel is cleared. The result of this operation is observable only if the Killed Pixel Mask Return flag is set on the input message.
1h	KEYFILTER_REPLACE_BLACK	In this mode, each texel that matches the chroma key is replaced with (0,0,0,0) (black with alpha=0) prior to filtering. For YCrCb surface formats, the black value is A=0, R(Cr)=0x80, G(Y)=0x10, B(Cb)=0x80. This will tend to darken/fade edges of keyed regions. Note that the pixel pipeline must be programmed to use the resulting filtered texel value to gain the intended effect, e.g., handle the case of a totally keyed-out region (filtered texel alpha==0) through use of alpha test, etc.

3:1 **Shadow Function**

Project:	All

SAMPLER_STATE

		Format: U3 Enumerated Type This field is used for shadow mapping support via the sample_c message type, and specifies the specific comparison operation to be used. The comparison is between the texture sample red channel (except for alpha-only formats which use the alpha channel), and the "ref" value provided in the input message.

Value	Name
0h	PREFILTEROP ALWAYS
1h	PREFILTEROP NEVER
2h	PREFILTEROP LESS
3h	PREFILTEROP EQUAL
4h	PREFILTEROP LEQUAL
5h	PREFILTEROP GREATER
6h	PREFILTEROP NOTEQUAL
7h	PREFILTEROP GEQUAL

	0	**Cube Surface Control Mode** Project: All Format: U1 Enumerated Type When sampling from a SURFTYPE_CUBE surface, this field controls whether the TC* Address Control Mode fields are interpreted as programmed or overridden to TEXCOORDMODE_CUBE.

Value	Name	Project
0h	PROGRAMMED	All
1h	OVERRIDE	All

Programming Notes
This field must be set to CUBECTRLMODE_PROGRAMMED

2	31:30	**Reserved** Project: CHV, BSW
	29:28	**Reserved** Project: CHV, BSW
	27:26	**Reserved** Project: CHV, BSW
	31:24	**Reserved** Project: CHV, BSW
	25:24	**Reserved** Project: CHV, BSW
	23:6	**Indirect State Pointer** Project: CHV, BSW

SAMPLER_STATE

Description	Project
This pointer is relative to the Dynamic State Base Address.	CHV, BSW

5 Reserved

Project:	CHV, BSW
Format:	MBZ

4 Reserved

Project:	CHV, BSW

3 Reserved

Project:	CHV, BSW

2 Reserved

Project:	CHV, BSW

1 Reserved

Project:	CHV, BSW

0 LOD Clamp Magnification Mode

Project:	CHV, BSW
Format:	U1 Enumerated Type

This field allows the flexibility to control how LOD clamping is handled when in magnification mode.

Value	Name	Description	Project
0h	MIPNONE	When in magnification mode, Sampler will clamp LOD as if the **Mip Mode Filter** is MIPFILTER_NONE. This is how OpenGL defines magnification, and therefore it is expected that those drivers would not set this bit.	All
1h	MIPFILTER	When in magnification mode, Sampler will clamp LOD based on the value of **Mip Mode Filter**.	All

3 31:24 Reserved

Project:	CHV, BSW

23:22 Reserved

Project:	CHV, BSW
Format:	MBZ

21:19 Maximum Anisotropy

Project:	All
Format:	U3 Enumerated Type

This field clamps the maximum value of the anisotropy ratio used by the MAPFILTER_ANISOTROPIC filter (Min or Mag Mode Filter).

Value	Name	Description	Project

SAMPLER_STATE

0h	RATIO 2:1	At most a 2:1 aspect ratio filter is used	All
1h	RATIO 4:1	At most a 4:1 aspect ratio filter is used	All
2h	RATIO 6:1	At most a 6:1 aspect ratio filter is used	All
3h	RATIO 8:1	At most a 8:1 aspect ratio filter is used	All
4h	RATIO 10:1	At most a 10:1 aspect ratio filter is used	All
5h	RATIO 12:1	At most a 12:1 aspect ratio filter is used	All
6h	RATIO 14:1	At most a 14:1 aspect ratio filter is used	All
7h	RATIO 16:1	At most a 16:1 aspect ratio filter is used	All

18 **U Address Mag Filter Rounding Enable**

Project:	All
Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes	Project
Hardware will **not** force rounding enable.	CHV, BSW

17 **U Address Min Filter Rounding Enable**

Project:	All
Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes	Project
Hardware will **not** force rounding enable.	CHV, BSW

16 **V Address Mag Filter Rounding Enable**

Project:	All
Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

Programming Notes	Project
Hardware will **not** force rounding enable.	CHV, BSW

15 **V Address Min Filter Rounding Enable**

Project:	All
Format:	Enable

Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode.

SAMPLER_STATE

		Programming Notes / **Project** Hardware will **not** force rounding enable. / CHV, BSW
	14	**R Address Mag Filter Rounding Enable** Project: All Format: Enable Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode. **Programming Notes** / **Project** Hardware will **not** force rounding enable. / CHV, BSW
	13	**R Address Min Filter Rounding Enable** Project: All Format: Enable Controls whether the texture address is rounded or truncated before being used to select texels to sample. Provides independent control of rounding on one texture address dimension (U/V/R) in either mag or min filter mode. **Programming Notes** / **Project** Hardware will **not** force rounding enable. / CHV, BSW
	12:11	**Trilinear Filter Quality** Project: All Format: U2 Enumerated Type Selects the quality level for the trilinear filter.

Value	Name	Description	Project
0	FULL	Full Quality. Both mip maps are sampled under all circumstances.	All
1	HIGH	High Quality. Same as full quality.	CHV, BSW
2	MED	Medium Quality. If the contribution of one mip map is less than 25%, only the other mip map contributes.	All
3	LOW	Low Quality. If the contribution of one mip map is less than 37.5%, only the other mip map contributes.	All

	10	**Non-normalized Coordinate Enable** Project: CHV, BSW Format: Enable This field, if enabled, specifies that the input coordinates (U/V/R) are in non-normalized space, where each integer increment is one texel on LOD 0. If disabled, coordinates are normalized, where the range 0 to 1 spans the entire surface. **Programming Notes** The following state must be set as indicated if this field is *enabled*:

SAMPLER_STATE

- TCX/Y/Z Address Control Mode must be TEXCOORDMODE_CLAMP, TEXCOORDMODE_HALF_BORDER, or TEXCOORDMODE_CLAMP_BORDER.
- Surface Type must be SURFTYPE_2D or SURFTYPE_3D.
- Mag Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR.
- Min Mode Filter must be MAPFILTER_NEAREST or MAPFILTER_LINEAR.
- Mip Mode Filter must be MIPFILTER_NONE.
- Min LOD must be 0.
- Max LOD must be 0.
- MIP Count must be 0.
- Surface Min LOD must be 0.
- Texture LOD Bias must be 0.

9 **Reserved**

Project:	CHV, BSW
Format:	MBZ

8:6 **TCX Address Control Mode**

Project:	All
Format:	Texture Coordinate Mode [CHV, BSW] Enumerated Type

Controls how the 1st (TCX, aka U) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). The setting of this field is subject to being overridden by the Cube Surface Control Mode field when sampling from a SURFTYPE_CUBE surface.

Programming Notes	Project
When using cube map texture coordinates, each TC component must have the same Address Control Mode.	CHV, BSW
When TEXCOORDMODE_CUBE is not used accessing a cube map, the map's Cube Face Enable field must be programmed to 111111b (all faces enabled).	
MAPFILTER_MONO: Texture addressing modes must all be set to TEXCOORDMODE_CLAMP_BORDER. The **Border Color** is ignored in this mode, a constant value of 0 is used for border color. Software must pad the border texels within the map itself with 0.	
If **Surface Format** is PLANAR*, this field must be set to TEXCOORDMODE_CLAMP.	

5:3 **TCY Address Control Mode**

Project:	All
Format:	Texture Coordinate Mode [CHV, BSW] Enumerated Type

Controls how the 2nd (TCY, aka V) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror). See Address TCX Control Mode above for details

Programming Notes

SAMPLER_STATE

		If this field is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER and a 1D surface is sampled, incorrect blending with the border color in the vertical direction may occur.
	2:0	**TCZ Address Control Mode** Project: All Format: Texture Coordinate Mode [CHV, BSW] Enumerated Type **Description** / **Project** Controls how the 3rd (TCZ) component of input texture coordinates are mapped to texture map addresses - specifically, how coordinates "outside" the texture are handled (wrap/clamp/mirror).See Address TCX Control Mode above for details If this field is set to TEXCOORDMODE_CLAMP_BORDER or TEXCOORDMODE_HALF_BORDER and a 3D surface is sampled, incorrect blending with the border color in the Q direction may occur. — CHV, BSW

SCISSOR_RECT

SCISSOR_RECT

Project: CHV, BSW
Source: RenderCS
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

The viewport-specific state used by the SF unit (SCISSOR_RECT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 2 DWords apart. The location of first element of the array, as specified by Pointer to SCISSOR_RECT, is aligned to a 32-byte boundary.

DWord	Bit	Description
0	31:16	**Scissor Rectangle Y Min** Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates less than Y Min will be clipped out if Scissor Rectangle is enabled. NOTE: If Y Min is set to a value greater than Y Max, all primitives will be discarded for this viewport. Value: [0,16383] — Name: — Project: CHV, BSW
	15:0	**Scissor Rectangle X Min** Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies X Min coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) X coordinates less than X Min will be clipped out if Scissor Rectangle is enabled. NOTE: If X Min is set to a value greater than X Max, all primitives will be discarded for this viewport. Value: [0,16383] — Name: — Project: CHV, BSW
1	31:16	**Scissor Rectangle Y Max** Project: All Format: U16 Pixels from Drawing Rectangle origin (upper left corner) Specifies Y Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than Y Max will be clipped out if Scissor Rectangle is enabled. Value: [0,16383] — Name: — Project: CHV, BSW
	15:0	**Scissor Rectangle X Max** Project: All

SCISSOR_RECT

Format:	U16 Pixels from Drawing Rectangle origin (upper left corner)

Specifies X Max coordinate of (inclusive) Scissor Rectangle used for scissor test. Pixels with (Draw Rectangle-relative) Y coordinates greater than X Max will be clipped out if Scissor Rectangle is enabled.

Value	Name	Project
0-16383		CHV, BSW

Scratch Hword Block Message Header

MH_A32_HWB - Scratch Hword Block Message Header

Project: CHV, BSW

Source: DataPort 0

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-2	95:0	**Reserved** Project: All Format: Ignore Ignored
3	31:0	**Per Thread Scratch Space** Project: All Format: MHC_PTSS [CHV, BSW] Specifies amount of scratch space used by this thread, for Stateless bounds checking.
4	31:0	**Reserved** Project: All Format: Ignore Ignored
5	31:0	**Buffer Base Address** Project: All Format: MHC_A32_BBA [CHV, BSW] Specifies the surface address offset page [31:10] for A32 stateless messages.
6-7	63:0	**Reserved** Project: All Format: Ignore Ignored

SF_CLIP_VIEWPORT

SF_CLIP_VIEWPORT

Project: CHV, BSW

Source: RenderCS

Size (in bits): 512

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Viewport Matrix Element m00** Format: IEEE_Float
1	31:0	**Viewport Matrix Element m11** Format: IEEE_Float
2	31:0	**Viewport Matrix Element m22** Format: IEEE_Float
3	31:0	**Viewport Matrix Element m30** Format: IEEE_Float
4	31:0	**Viewport Matrix Element m31** Format: IEEE_Float
5	31:0	**Viewport Matrix Element m32** Format: IEEE_Float
6	31:0	**Reserved** Format: MBZ
7	31:0	**Reserved** Format: MBZ
8	31:0	**X Min Clip Guardband** Format: IEEE_Float . This 32-bit float represents the XMin guardband boundary (normalized to Viewport.XMin == -1.0f). This corresponds to the left boundary of the NDC guardband. Workaround: Minimum allowed value for this field is -16384. — Project: CHV, BSW
9	31:0	**X Max Clip Guardband** Format: IEEE_Float This 32-bit float represents the XMax guardband boundary (normalized to Viewport..XMax == 1.0f). This corresponds to the right boundary of the NDC guardband. Workaround — Project

SF_CLIP_VIEWPORT

		Maximum allowed value for this field is 16383. — CHV, BSW
10	31:0	**Y Min Clip Guardband** Format: — IEEE_Float This 32-bit float represents the YMin guardband boundary (normalized to Viewport.YMin == -1.0f). This corresponds to the bottom boundary of the NDC guardband. **Workaround** / **Project** Minimum allowed value for this field is -16384. — CHV, BSW
11	31:0	**Y Max Clip Guardband** Format: — IEEE_Float This 32-bit float represents the YMax guardband boundary (normalized to Viewport.YMax == 1.0f). This corresponds to the top boundary of the NDC guardband. **Workaround** / **Project** Maximum allowed value for this field is 16383. — CHV, BSW
12 **Project:** CHV, BSW	31:0	**X Min ViewPort** Project: — CHV, BSW Format: — IEEE_Float This 32-bit float represents the Viewport.XMin. This is the X min of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
13 **Project:** CHV, BSW	31:0	**X Max ViewPort** Project: — CHV, BSW Format: — IEEE_Float This 32-bit float represents the Viewport.XMax. This is the X max of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
14 **Project:** CHV, BSW	31:0	**Y Min ViewPort** Project: — CHV, BSW Format: — IEEE_Float This 32-bit float represents the Viewport.YMin. This is the Y min of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.
15 **Project:** CHV, BSW	31:0	**Y Max ViewPort** Project: — CHV, BSW Format: — IEEE_Float

SF_CLIP_VIEWPORT		
		This 32-bit float represents the Viewport.Ymax. This is the Y max of the viewport extents as programmed by API, and this value should be programmed in Screen Space coordinate and not as normalized coordinate.

SF_OUTPUT_ATTRIBUTE_DETAIL

SF_OUTPUT_ATTRIBUTE_DETAIL

Source: RenderCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15	**Component Override W** Format: Enable If set, the W component of this output Attribute is overridden by the W component of the constant vector specified by ConstantSource.
	14	**Component Override Z** Format: Enable If set, the Z component of this output Attribute is overridden by the Z component of the constant vector specified by ConstantSource.
	13	**Component Override Y** Format: Enable If set, the Y component of output Attribute is overridden by the Y component of the constant vector specified by ConstantSource.
	12	**Component Override X** Format: Enable If set, the X component of output Attribute is overridden by the X component of the constant vector specified by ConstantSource.
	11	**Swizzle Control Mode** Project: CHV, BSW Format: U1 Enumerated Type When Attribute Swizzle Enable is ENABLED, this bit controls whether attributes 0-15 or 16-31 are subject to the following swizzle controls: • Component Override X/Y/Z/W • Constant Source • Swizzle Select • Source Attribute • WrapShortest Enables Note that the Number of SF Output Attributes field specifies how many attributes are output. Note: This field does not impact any functions which provide separate states for all 32 attributes (e.g., Point sprite, Constant interpolation).

SF_OUTPUT_ATTRIBUTE_DETAIL

Note: This field is only valid for the first indexed attribute (Attribute[0]). For all other indices, it is Reserved and MBZ.

10:9 Constant Source

Format:	U2 enumerated type

This state selects a constant vector which can be used to override individual components of this Attribute

Value	Name	Description
0h	CONST_0000	Constant.xyzw = 0.0,0.0,0.0,0.0
1h	CONST_0001_FLOAT	Constant.xyzw = 0.0,0.0,0.0,1.0
2h	CONST_1111_FLOAT	Constant.xyzw = 1.0,1.0,1.0,1.0
3h	PRIM_ID	Constant.xyzw = PrimID (replicated)

8 Reserved

Format:	MBZ

7:6 Swizzle Select

Format:	U2 enumerated type

This state, along with Source Attribute, specifies the source for this output Attribute.

Value	Name	Description
0h	INPUTATTR	This attribute is sourced from AttrInputReg[SourceAttribute]
1h	INPUTATTR_FACING	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1].
2h	INPUTATTR_W	This attribute is sourced from AttrInputReg[SourceAttribute]. The W component is copied to the X component.
3h	INPUTATTR_FACING_W	If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. If the object is back-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. The W component is copied to the X component.

5 Reserved

Format:	MBZ

4:0 Source Attribute

Format:	U5

This field selects the source attribute for this Attribute. Source attribute 0 corresponds to the first 128 bits of data indicated by Vertex URB Entry Read Offset

SFC_8x8_AVS_COEFFICIENTS

SFC_8x8_AVS_COEFFICIENTS

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

Description	Project
ExistsIf = AVS	CHV, BSW

DWord	Bit	Description
0	31:24	**ZeroYFilterCoefficient1** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient1** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient0** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient0** Format: S1.6 2's Complement **Range:** [-2, +2)
1	31:24	**ZeroYFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-2, +2)

SFC_8x8_AVS_COEFFICIENTS		
	7:0	**ZeroXFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-2, +2)
2	31:24	**ZeroYFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2, +2)
3	31:24	**ZeroYFilterCoefficient7** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**ZeroXFilterCoefficient7** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**ZeroYFilterCoefficient6** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**ZeroXFilterCoefficient6** Format: S1.6 2's Complement **Range:** [-2, +2)
4	31:24	**OneXFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SFC_8x8_AVS_COEFFICIENTS		
	23:16	**OneXFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	15:0	**Reserved** Format: MBZ
5	31:16	**Reserved** Format: MBZ
	15:8	**OneXFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	7:0	**OneXFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
6	31:24	**OneYFilterCoefficient3** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)
	23:16	**OneYFilterCoefficient2** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	15:0	**Reserved** Format: MBZ
7	31:16	**Reserved** Format: MBZ
	15:8	**OneYFilterCoefficient5** Format: S1.6 2's Complement **Range:** [-1.0, +1.0)
	7:0	**OneYFilterCoefficient4** Format: S1.6 2's Complement **Range:** [-2.0, +2.0)

SIMD4x2 Typed Surface 32-Bit Address Payload

MAP32B_TS_SIMD4X2 - SIMD4x2 Typed Surface 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1	31:0	**V0** Format: U32 Specifies the V channel address offset for slot 0.
2	31:0	**R0** Format: U32 Specifies the R channel address offset for slot 0.
3	31:0	**LOD0** Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 0.
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5	31:0	**V1** Format: U32 Specifies the V channel address offset for slot 1.
6	31:0	**R1** Format: U32 Specifies the R channel address offset for slot 1.
7	31:0	**LOD1**

MAP32B_TS_SIMD4X2 - SIMD4x2 Typed Surface 32-Bit Address Payload

		Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 1.

SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD4X2 - SIMD4x2 Untyped BUFFER Surface 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1-3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5-7	95:0	**Reserved** Format: Ignore Ignored

SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD4X2 - SIMD4x2 Untyped BUFFER Surface 64-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**U0** Format: U64 Specifies the U channel address offset for slot 0.
2-3	63:0	**Reserved** Format: Ignore Ignored
4-5	63:0	**U1** Format: U64 Specifies the U channel address offset for slot 1.
6-7	63:0	**Reserved** Format: Ignore Ignored

SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD4X2 - SIMD4x2 Untyped STRBUF Surface 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**U0** Format: U32 Specifies the U channel address offset for slot 0.
1	31:0	**V0** Format: U32 Specifies the V channel address offset for slot 0.
2-3	63:0	**Reserved** Format: Ignore Ignored
4	31:0	**U1** Format: U32 Specifies the U channel address offset for slot 1.
5	31:0	**V1** Format: U32 Specifies the V channel address offset for slot 1.
6-7	63:0	**Reserved** Format: Ignore Ignored

SIMD4x2 32-Bit Address Payload

MAP32B_SIMD4X2 - SIMD4x2 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**Offset0** Format: U32 Specifies the address offset for slot 0.
1-3	95:0	**Reserved** Format: Ignore Ignored
4	31:0	**Offset1** Format: U32 Specifies the address offset for slot 1.
5-7	95:0	**Reserved** Format: Ignore Ignored

SIMD8 Dual Source Render Target Data Payload

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Red
1.0-1.7	255:0	**Src0 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Green
2.0-2.7	255:0	**Src0 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Blue
3.0-3.7	255:0	**Src0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Alpha
4.0-4.7	255:0	**Src1 Red** Project: All

MDP_RTW_8DS - SIMD8 Dual Source Render Target Data Payload

		Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Red
5.0-5.7	255:0	**Src1 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Green
6.0-6.7	255:0	**Src1 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Alpha

SIMD8 LOD Message Address Payload Control

MACR_LOD_SIMD8 - SIMD8 LOD Message Address Payload Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Slot0 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 0
0.1	31:0	**Slot1 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 1
0.2	31:0	**Slot2 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 2
0.3	31:0	**Slot3 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 3
0.4	31:0	**Slot4 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 4
0.5	31:0	**Slot5 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 5

MACR_LOD_SIMD8 - SIMD8 LOD Message Address Payload Control

0.6	31:0	**Slot6 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 6
0.7	31:0	**Slot7 LOD** Project: All Format: MACD_LOD [CHV, BSW] Specifies the LOD for slot 7

SIMD8 Render Target Data Payload

MDP_RTW_8 - SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
1.0-1.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
2.0-2.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

SIMD8 Typed Surface 32-Bit Address Payload

MAP32B_TS_SIMD8 - SIMD8 Typed Surface 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**V** Project: All Format: MACR_32b [CHV, BSW] Specifies the V channel for slots [7:0]
2.0-2.7	255:0	**R** Project: All Format: MACR_32b [CHV, BSW] Specifies the R channel for slots [7:0]
3.0-3.7	255:0	**LOD** Project: All Format: MACR_LOD_SIMD8 [CHV, BSW] Specifies the LOD for slots [7:0]

SIMD8 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 32-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [7:0]

SIMD8 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD8 - SIMD8 Untyped BUFFER Surface 64-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U3_U0** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [3:0]
1.0-1.7	255:0	**U7_U4** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [7:4]

SIMD8 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD8 - SIMD8 Untyped STRBUF Surface 32-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**V** Project: All Format: MACR_32b [CHV, BSW] Specifies the V channel for slots [7:0]

SIMD16 Render Target Data Payload

MDP_RTW_16 - SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2048
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
1.0-1.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red
2.0-2.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Green[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green
4.0-4.7	255:0	**Blue[7:0]** Project: All

MDP_RTW_16 - SIMD16 Render Target Data Payload

		Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
5.0-5.7	255:0	**Blue[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
6.0-6.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
7.0-7.7	255:0	**Alpha[15:7]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:7] Alpha

SIMD16 Untyped BUFFER Surface 32-Bit Address Payload

MAP32B_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 32-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U[7:0]** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**U[15:8]** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [15:8]

SIMD16 Untyped BUFFER Surface 64-Bit Address Payload

MAP64B_USU_SIMD16 - SIMD16 Untyped BUFFER Surface 64-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U3_U0** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [3:0]
1.0-1.7	255:0	**U7_U4** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [7:4]
2.0-2.7	255:0	**U11_U8** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [11:8]
3.0-3.7	255:0	**U15_U12** Project: All Format: MACR_64b [CHV, BSW] Specifies the U channel for slots [15:12]

SIMD16 Untyped STRBUF Surface 32-Bit Address Payload

MAP32B_USUV_SIMD16 - SIMD16 Untyped STRBUF Surface 32-Bit Address Payload

Project:	All
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**U7_U0** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [7:0]
1.0-1.7	255:0	**U15_U8** Project: All Format: MACR_32b [CHV, BSW] Specifies the U channel for slots [15:8]
2.0-2.7	255:0	**V7_V0** Project: All Format: MACR_32b [CHV, BSW] Specifies the V channel for slots [7:0]
3.0-3.7	255:0	**V15_V8** Project: All Format: MACR_32b [CHV, BSW] Specifies the V channel for slots [15:8]

SIMD 32-Bit Address Payload Control

MACR_32B - SIMD 32-Bit Address Payload Control

Project: CHV, BSW
Source: PRM
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Offset0** Project: All Format: U32 Specifies the address offset for slot 0 in this payload register.
0.1	31:0	**Offset1** Project: All Format: U32 Specifies the address offset for slot 1 in this payload register.
0.2	31:0	**Offset2** Project: All Format: U32 Specifies the address offset for slot 2 in this payload register.
0.3	31:0	**Offset3** Project: All Format: U32 Specifies the address offset for slot 3 in this payload register.
0.4	31:0	**Offset4** Project: All Format: U32 Specifies the address offset for slot 4 in this payload register.
0.5	31:0	**Offset5** Project: All Format: U32 Specifies the address offset for slot 5 in this payload register.

MACR_32B - SIMD 32-Bit Address Payload Control		
0.6	31:0	**Offset6** Project: All Format: U32 Specifies the address offset for slot 6 in this payload register.
0.7	31:0	**Offset7** Project: All Format: U32 Specifies the address offset for slot 7 in this payload register.

SIMD 64-Bit Address Payload Control

MACR_64B - SIMD 64-Bit Address Payload Control

Project: CHV, BSW
Source: PRM
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.1	63:0	**Offset0** Project: All Format: U64 Specifies the address offset for slot 0 in this payload register.
0.2-0.3	63:0	**Offset1** Project: All Format: U64 Specifies the address offset for slot 1 in this payload register.
0.4-0.5	63:0	**Offset2** Project: All Format: U64 Specifies the address offset for slot 2 in this payload register.
0.6-0.7	63:0	**Offset3** Project: All Format: U64 Specifies the address offset for slot 3 in this payload register.

SIMD8 32-Bit Address Payload

MAP32B_SIMD8 - SIMD8 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[7:0]** Project: All Format: MACR_32b [CHV, BSW] Specifies the address offset for Slots [7:0].

SIMD8 64-Bit Address Payload

MAP64B_SIMD8 - SIMD8 64-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[3:0]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offset for slots [3:0].
1.0-1.7	255:0	**Offset[7:4]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offset for slots [7:4].

SIMD16 32-Bit Address Payload

MAP32B_SIMD16 - SIMD16 32-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	512
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[7:0]** Project: All Format: MACR_32b [CHV, BSW] Specifies the address offset for slots [7:0].
1.0-1.7	255:0	**Offset[15:8]** Project: All Format: MACR_32b [CHV, BSW] Specifies the address offset for slots [15:8].

SIMD16 64-Bit Address Payload

MAP64B_SIMD16 - SIMD16 64-Bit Address Payload

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1024
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Offset[3:0]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offsets for slots [3:0].
1.0-1.7	255:0	**Offset[7:4]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offsets for slots [7:4].
2.0-2.7	255:0	**Offset[11:8]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offsets for slots [11:8].
3.0-3.7	255:0	**Offset[15:12]** Project: All Format: MACR_64b [CHV, BSW] Specifies the address offsets for slots [15:12].

SIMD Mode 2 Message Descriptor Control Field

MDC_SM2 - SIMD Mode 2 Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	1
Default Value:	0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Project: All Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description	Project
00h	SIMD8	SIMD8	All
01h	SIMD16	SIMD16	All

SIMD Mode 3 Message Descriptor Control Field

MDC_SM3 - SIMD Mode 3 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**SIMD Mode** Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description	Project
00h	SIMD4x2	SIMD4x2	
01h	SIMD16	SIMD16	
02h	SIMD8	SIMD8	
03h	Reserved	Ignored	

SLM Surface Pixel Mask Message Header

MH1_SLM_PSM - SLM Surface Pixel Mask Message Header

Project:	CHV, BSW
Source:	DataPort 1
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0-6	223:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

Slot Group 2 Message Descriptor Control Field

MDC_SG2 - Slot Group 2 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Project: All Format: Enumeration Controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present. Value / Name / Description / Project 00h / SG8L / Use low 8 slots / All 01h / SG8U / Use high 8 slots / All

Slot Group 3 Message Descriptor Control Field

MDC_SG3 - Slot Group 3 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 2
Default Value: 0x00000000

DWord	Bit	Description
0	1:0	**SIMD Mode** Format: Enumeration Controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present.

Value	Name	Description	Project
00h	SG4x2	SIMD4x2	
01h	SG8L	Use low 8 slots	
02h	SG8U	Use high 8 slots	
03h	Reserved	Ignored	

Slot Group Select Render Cache Message Descriptor Control Field

MDC_RT_SGS - Slot Group Select Render Cache Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**Slot Group Select** Project: All This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the antialias alpha, multisample coverage mask, and if the header is not present also includes the X/Y addresses and pixel enables. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed.

Value	Name	Description	Project
00h	SLOTGRP_LO	Choose bypassed data for slots 15:0	All
01h	SLOTGRP_HI	Choose bypassed data for slots 31:16	All

SO_DECL

SO_DECL

Project:	CHV, BSW
Source:	RenderCS
Size (in bits):	16
Default Value:	0x00000000

A list of SO_DECL structures are passed in the 3DSTATE_SO_DECL_LIST command. Each structure specifies either (a) the source and destination of an up-to-4-DWord appending write into an SO buffer, or (b) how many DWords to skip over in the destination SO buffer (i.e., a "hole" where the previous buffer contents are maintained).

DWord	Bit	Description
0	15:14	**Reserved** Project: All Format: MBZ
	13:12	**Output Buffer Slot** Project: All Format: U2 Buffer Index This field selects the destination output buffer slot.
	11	**Hole Flag** Project: All Format: Flag If set, the Component Mask field indirectly specifies a number of 32-bit locations to skip over (leave unmodified in memory) in the selected output buffer. The Register Index field is ignored. The only permitted Component Mask values are as follows: 0x0 No Dwords are skipped over (SO_DECL performs no operation) 0x1 (X) Skip 1 DWord 0x3 (XY) Skip 2 DWords 0x7 (XYZ) Skip 3 DWords 0xF (XYZW) Skip 4 DWords
	10	**Reserved** Project: All Format: MBZ
	9:4	**Register Index** Project: All Format: U6 128-bit granular offset into the source vertex read data

SO_DECL

If Hole Flag is clear, this field specifies the 128-bit offset into the source vertex data which supplies the source data to be written to the destination buffer, where the individual 32-component destination locations are selected by Component Mask. e.g., Register Index 0 corresponds with the first 128 bits of the data read from the vertex URB entry (as per corresponding Vertex Read Offset state)

There is only enough internal storage for the 128-bit vertex header and 32 128-bit vertex attributes.

Value	Name
[0,32]	
0h	**[Default]**

Programming Notes
It is the responsibility of software to map any API-visible source data specifications (e.g., vertex register number) into 128-bit granular URB read offsets.

3:0 **Component Mask**

Project:	All
Format:	MASK 4-bit Mask

This field is a 4-bit bitmask that selects which contiguous 32-bit component(s) are either written or skipped-over in the destination buffer. If this field is zero the SO_DECL operation is effectively a no-op. No data will be appended to the destination and the destination buffer's write pointer will not be advanced. If the **Hole Flag** is set, this field (if non-zero) indirectly specifies how much the destination buffer's write pointer should be advanced.See **Hole Flag** description above for restrictions on this field. If the **Hole Flag** is clear, this field (if non-zero) selects which source components are to be written to the destination buffer. The components must be contiguous, e.g. YZW is legal, but XZW is not. The selected source components are written to the destination buffer starting at the current write pointer, and then the write pointer is advanced past the written data. E.g., if YZW is specified, the three (YZW) components of the source register will be written to the destination buffer at the current write pointer, and the write pointer will be advanced by 3 DWords.

Value	Name	Project
0h	**[Default]**	
xxx1b	SO_DECL_COMPMASK_X	All
xx1xb	SO_DECL_COMPMASK_Y	All
x1xxb	SO_DECL_COMPMASK_Z	All
1xxxb	SO_DECL_COMPMASK_W	All

SO_DECL_ENTRY

SO_DECL_ENTRY		
Project:	CHV, BSW	
Source:	RenderCS	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0..1	63:48	**Stream 3 Decl** Format: SO_DECL [CHV, BSW] This field contains Stream 3 SO_DECL [n]
	47:32	**Stream 2 Decl** Format: SO_DECL [CHV, BSW] This field contains Stream 2 SO_DECL [n]
	31:16	**Stream 1 Decl** Format: SO_DECL [CHV, BSW] This field contains Stream 1 SO_DECL [n]
	15:0	**Stream 0 Decl** Format: SO_DECL [CHV, BSW] This field contains Stream 0 SO_DECL [n]

SplitBaseAddress4KByteAligned

SplitBaseAddress4KByteAligned		
Source:	PRM	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Specifies a 64-bit (48-bit canonical) 4K-byte aligned memory base address.		
DWord	**Bit**	**Description**
0 **Project:** All	31:12	**Base Address Low** Project: All Format: GraphicsAddress[31:12]
	11:0	**Reserved** Project: All Format: MBZ
1 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Format: MBZ

SplitBaseAddress64ByteAligned

SplitBaseAddress64ByteAligned		
Source:	PRM	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Specifies a 64-bit (48-bit canonical) 64-byte aligned memory base address.		
DWord	**Bit**	**Description**
0 **Project:** All	31:6	**Base Address Low** Project: All Format: GraphicsAddress[31:6]
	5:0	**Reserved** Project: All Format: MBZ
1 **Project:** CHV, BSW	31:0	**Reserved** Project: CHV, BSW Format: MBZ

SrcRegNum

SrcRegNum

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	8
Default Value:	0x00000000

Description	Project
Register Number The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number. An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers. RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands.	CHV, BSW

DWord	Bit	Description
0	7:0	**Source Register Number** Value / Name / Description 0-127 / If {Dst/Src0/Src1/Src2}.RegFile==GRF / 0-0ffh / If {Dst/Src0/Src1/Src2}.RegFile==ARF / This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

SrcSubRegNum

SrcSubRegNum

Project:	CHV, BSW
Source:	Eulsa
Size (in bits):	5
Default Value:	0x00000000

Description	Project
Subregister Number The subregister number for the operand. For a GRF register, is the byte address within a 256-bit (32-byte) register. For an ARF register, determines the sub-register number according to the specified encoding for the given architecture register. RegNum and SubRegNum together provide the byte-aligned address for the origin of a GRF register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero.	CHV, BSW

Programming Notes
Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

DWord	Bit	Description
0	4:0	**Source Sub Register Number**

Value	Name	Description
0-31	If {Dst/Src0/Src1/Src2}.RegFile==GRF	
0-0ffh	If {Dst/Src0/Src1/Src2}.RegFile==ARF	This field is used to encode the architecture register as well as providing the register number. See GEN Execution Environment chapter for details.

Stateless Binding Table Index Message Descriptor Control Field

MDC_STATELESS - Stateless Binding Table Index Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 8
Default Value: 0x000000FF

DWord	Bit	Description
0	7:0	**Binding Table Index** Project: All Format: Enumeration Specifies the message is Stateless

Value	Name	Description	Project
0FFh	A32_A64 **[Default]**	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)	All
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).	All
Others	Reserved	Ignored	All

Restriction
When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Stateless Block Message Header

MH_A32_GO - Stateless Block Message Header

Project: CHV, BSW
Source: DataPort 0
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-1	63:0	**Reserved** Project: All Format: Ignore Ignored
2	31:0	**Global Offset** Project: All Format: U32 Specifies the global element index into the buffer, in units of Owords, Dwords, or Bytes (depending on the message). **Programming Notes** If the address offset calculated with the Buffer Base Address and Global Offset is greater than the PTSS size or the GeneralStateBufferSize, then the access is Out-of-Bounds.
3	31:0	**Per Thread Scratch Space** Project: All Format: MHC_PTSS [CHV, BSW] Specifies amount of scratch space used by this thread, for Stateless bounds checking.
4	31:0	**Reserved** Project: All Format: Ignore Ignored
5	31:0	**Buffer Base Address** Project: All Format: MHC_A32_BBA [CHV, BSW] Specifies the surface address offset page [31:10] for A32 stateless messages.
6-7	63:0	**Reserved** Project: All

MH_A32_GO - Stateless Block Message Header			
		Format:	Ignore
		Ignored	

Stateless Surface Message Header

MH1_A32 - Stateless Surface Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0-4	159:0	**Reserved** Project: All Format: Ignore Ignored
5	31:0	**Buffer Base Address** Project: All Format: MHC_A32_BBA [CHV, BSW] Specifies the surface address offset page [31:10] for A32 stateless messages.
6-7	63:0	**Reserved** Project: All Format: Ignore Ignored

Stateless Surface Pixel Mask Message Header

MH1_A32_PSM - Stateless Surface Pixel Mask Message Header

Project:	CHV, BSW
Source:	DataPort 1
Size (in bits):	256
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0-4	159:0	**Reserved** Format: Ignore Ignored
5	31:0	**Buffer Base Address** Format: MHC_A32_BBA [CHV, BSW] Specifies the surface address offset page [31:10] for A32 stateless messages.
6	31:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Project: Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

Subset Atomic Integer Trinary Operation Message Descriptor Control Field

MDC_AOP3S - Subset Atomic Integer Trinary Operation Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x0000000E

DWord	Bit	Description
0	3:0	**Atomic Integer Operation Type** Project: All Format: Enumeration Specifies the atomic integer trinary operation to be performed

Value	Name	Description	Project
0Eh	AOP_CMPWR **[Default]**	new_dst = (src0 == old_dst) ? src1 : old_dst	All
Others	Reserved	Ignored	All

Programming Notes
When Return Data Control is set, old_dst is returned.

Subset Reversed SIMD Mode 2 Message Descriptor Control Field

MDC_SM2RS - Subset Reversed SIMD Mode 2 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000001

DWord	Bit	Description
0	0	**SIMD Mode** Project: All Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description	Project
0h	Reserved	Not used	All
01h	SIMD8 **[Default]**	SIMD8	All

Subset SIMD Mode 2 Message Descriptor Control Field

MDC_SM2S - Subset SIMD Mode 2 Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 1
Default Value: 0x00000000

DWord	Bit	Description
0	0	**SIMD Mode** Project: All Format: Enumeration Specifies the SIMD mode of the message (number of slots processed)

Value	Name	Description	Project
00h	SIMD8	SIMD8	All
01h	Reserved	Ignored	All

Subset SIMD Mode 3 Message Descriptor Control Field

MDC_SM3S - Subset SIMD Mode 3 Message Descriptor Control Field

Project:	CHV, BSW
Source:	PRM
Size (in bits):	2
Default Value:	0x00000000

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>0</td><td>1:0</td><td>SIMD Mode
<table>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Format:</td><td>Enumeration</td></tr>
</table>
Specifies the SIMD mode of the message (number of slots processed)
<table>
<tr><th>Value</th><th>Name</th><th>Description</th><th>Project</th></tr>
<tr><td>00h</td><td>SIMD4x2</td><td>SIMD4x2</td><td></td></tr>
<tr><td>01h</td><td>Reserved</td><td>Ignored</td><td></td></tr>
<tr><td>02h</td><td>SIMD8</td><td>SIMD8</td><td></td></tr>
<tr><td>03h</td><td>Reserved</td><td>Ignored</td><td></td></tr>
</table>
</td></tr>
</table>

Subspan Render Target Message Header Control

MHC_RT_SUBSPAN - Subspan Render Target Message Header Control

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:16	**Y** Project: All Format: U16 Y coordinate for upper-left pixel of this subspan
	15:0	**X** Project: All Format: U16 X coordinate for upper-left pixel of this subspan

Surface Binding Table Index Message Descriptor Control Field

MDC_BTS - Surface Binding Table Index Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Project: All Format: Enumeration Specifies the Binding Table index for the message, which must be a Surface State Model.

Value	Name	Description	Project
00h-0EFh	BTS	Index of Binding Table State Surfaces	All
F0h-0FBh	Reserved	Reserved for future use	All
0FCh	Reserved	Reserved for future use	CHV, BSW
Others	Reserved	Ignored	All

Surface or Stateless Binding Table Index Message Descriptor Control Field

MDC_BTS_A32 - Surface or Stateless Binding Table Index Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 8
Default Value: 0x00000000

DWord	Bit	Description
0	7:0	**Binding Table Index** Project: All Format: Enumeration Specifies the surface for the message, either Surface State Model or Stateless.

Value	Name	Description	Project
00h-0EFh	BTS	Index of Binding Table State Surfaces	All
F0h-0FBh	Reserved	Reserved for future use	All
0FCh	Reserved	Reserved for future use	CHV, BSW
0FFh	A32_A64	Specifies a A32 or A64 Stateless access that is locally coherent (coherent within a thread group)	All
0FDh	A32_A64_NC	Specifies a A32 or A64 Stateless access that is non-coherent (coherent within a thread).	All
Others	Reserved	Ignored	All

Restriction
When using A32_A64_NC, SW must ensure that 2 threads do not both access the same cache line (64B)

Surface Pixel Mask Message Header

MH1_BTS_PSM - Surface Pixel Mask Message Header

Project: CHV, BSW

Source: DataPort 1

Size (in bits): 256

Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0000FFFF

DWord	Bit	Description
0-6	223:0	**Reserved** Format: Ignore Ignored
7	31:0	**Pixel Sample Mask** Project: Format: MHC_PSM Specifies the 16-bit Pixel/Sample Mask used with SIMD16 and SIMD8 surfaces.

SW Generated BINDING_TABLE_STATE

SW Generated BINDING_TABLE_STATE

Project: CHV, BSW
Source: PRM
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:5	**Surface State Pointer** Format: SurfaceStateOffset[31:5] This 32-byte aligned address points to a surface state block. This pointer is relative to the **Surface State Base Address** **Programming Notes** / **Project** Bit 5 of this pointer must be zero (i.e. **Surface State Pointer** must be 64-byte aligned). / CHV, BSW
	4:0	**Reserved** Format: MBZ

SZ OM S0A SIMD8 Render Target Data Payload

MDP_RTW_ZMA8 - SZ OM S0A SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1792
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [7:0] oMask. Upper half ignored.
2.0-2.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
3.0-3.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
4.0-4.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW]

MDP_RTW_ZMA8 - SZ OM S0A SIMD8 Render Target Data Payload

		Slots [7:0] Blue
5.0-5.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
6.0-6.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth

SZ OM S0A SIMD16 Render Target Data Payload

MDP_RTW_ZMA16 - SZ OM S0A SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	3328
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-1.7	511:0	**Source 0 Alpha** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Source 0 Alpha
2.0-2.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [15:0] oMask
3.0-4.7	511:0	**Red** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Red

MDP_RTW_ZMA16 - SZ OM S0A SIMD16 Render Target Data Payload		
5.0-6.7	511:0	**Green** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Green
7.0-8.7	511:0	**Blue** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Blue
9.0-10.7	511:0	**Alpha** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Alpha
11.0-12.7	511:0	**Source Depth** Project: All Format: MDP_DW_SIMD16 [CHV, BSW] Slots [15:0] Source Depth

SZ OM SIMD8 Dual Source Render Target Data Payload

MDP_RTW_ZM8DS - SZ OM SIMD8 Dual Source Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2560
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] oMask for slots [7:0] and [15:8]. Operation selects upper or lower half.
1.0-1.7	255:0	**Src0 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Red
2.0-2.7	255:0	**Src0 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Green
3.0-3.7	255:0	**Src0 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Blue

MDP_RTW_ZM8DS - SZ OM SIMD8 Dual Source Render Target Data Payload

4.0-4.7	255:0	**Src0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Alpha
5.0-5.7	255:0	**Src1 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Red
6.0-6.7	255:0	**Src1 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Green
7.0-7.7	255:0	**Src1 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Blue
8.0-8.7	255:0	**Src1 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Alpha
9.0-9.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] or [15:8] of Source Depth

SZ OM SIMD8 Render Target Data Payload

MDP_RTW_ZM8 - SZ OM SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [7:0] oMask. Upper half ignored.
1.0-1.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

MDP_RTW_ZM8 - SZ OM SIMD8 Render Target Data Payload		
5.0-5.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth

SZ OM SIMD16 Render Target Data Payload

MDP_RTW_ZM16 - SZ OM SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2816
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**oMask** Project: All Format: MDPR_OMASK [CHV, BSW] Slots [15:0] oMask
1.0-1.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red
3.0-3.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green

MDP_RTW_ZM16 - SZ OM SIMD16 Render Target Data Payload

4.0-4.7	255:0	**Green[15:7]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green
5.0-5.7	255:0	**Blue[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
6.0-6.7	255:0	**Blue[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
7.0-7.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
8.0-8.7	255:0	**Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Alpha
9.0-9.7	255:0	**Source Depth[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth
10.0-10.7	255:0	**Source Depth[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Source Depth

SZ S0A SIMD8 Render Target Data Payload

MDP_RTW_ZA8 - SZ S0A SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1536
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
2.0-2.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
4.0-4.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha

MDP_RTW_ZA8 - SZ S0A SIMD8 Render Target Data Payload

5.0-5.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth

SZ S0A SIMD16 Render Target Data Payload

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	3072
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Source 0 Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source 0 Alpha
1.0-1.7	255:0	**Source 0 Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Source 0 Alpha
2.0-2.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
3.0-3.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW]

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

		Slots [15:8] Red
4.0-4.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
5.0-5.7	255:0	**Green[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green
6.0-6.7	255:0	**Blue[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
7.0-7.7	255:0	**Blue[15:7]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
8.0-8.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
9.0-9.7	255:0	**Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Alpha
10.0-10.7	255:0	**Source Depth[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth
11.0-11.7	255:0	**Source Depth[15:8]** Project: All

MDP_RTW_ZA16 - SZ S0A SIMD16 Render Target Data Payload

		Format:	MDP_DW_SIMD8 [CHV, BSW]
		Slots [15:8] Source Depth	

SZ SIMD8 Dual Source Render Target Data Payload

MDP_RTW_Z8DS - SZ SIMD8 Dual Source Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2304
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Src0 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Red
1.0-1.7	255:0	**Src0 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Green
2.0-2.7	255:0	**Src0 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Blue
3.0-3.7	255:0	**Src0 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src0 Alpha

MDP_RTW_Z8DS - SZ SIMD8 Dual Source Render Target Data Payload

4.0-4.7	255:0	**Src1 Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Red
5.0-5.7	255:0	**Src1 Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Green
6.0-6.7	255:0	**Src1 Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Blue
7.0-7.7	255:0	**Src1 Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots[7:0] or [15:8] of Src1 Alpha
8.0-8.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] or [15:8] of Source Depth

SZ SIMD8 Render Target Data Payload

MDP_RTW_Z8 - SZ SIMD8 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	1280
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
1.0-1.7	255:0	**Green** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
2.0-2.7	255:0	**Blue** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
3.0-3.7	255:0	**Alpha** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
4.0-4.7	255:0	**Source Depth** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth

SZ SIMD16 Render Target Data Payload

MDP_RTW_Z16 - SZ SIMD16 Render Target Data Payload

Project:	All
Source:	PRM
Size (in bits):	2560
Default Value:	0x00000000, 0x00000000

DWord	Bit	Description
0.0-0.7	255:0	**Red[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Red
1.0-1.7	255:0	**Red[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Red
2.0-2.7	255:0	**Green[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Green
3.0-3.7	255:0	**Green[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Green

MDP_RTW_Z16 - SZ SIMD16 Render Target Data Payload		
4.0-4.7	255:0	**Blue[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Blue
5.0-5.7	255:0	**Blue[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Blue
6.0-6.7	255:0	**Alpha[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Alpha
7.0-7.7	255:0	**Alpha[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Alpha
8.0-8.7	255:0	**Source Depth[7:0]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [7:0] Source Depth
9.0-9.7	255:0	**Source Depth[15:8]** Project: All Format: MDP_DW_SIMD8 [CHV, BSW] Slots [15:8] Source Depth

Thread Spawn Message Descriptor

Thread Spawn Message Descriptor

Project: All
Source: RenderCS
Size (in bits): 32
Default Value: 0x00000000

DWord	Bit	Description
0	31:20	**Reserved** Format: MBZ
	19	**Header Present** Format: MBZ **Programming Notes** This bit MBZ for all Thread Spawner messages.
	18:5	**Reserved** Format: MBZ
	4	**Resource Select** This field specifies the resource associated with the action taken by the Opcode. Value / Name / Description / Exists If: 0 / Spawn Child / Spawn a Child Thread / [Opcode] == 'Spawn Thread' 1 / Spawn Root / Spawn a Root Thread / [Opcode] == 'Spawn Thread' 0 / Dereference Resource / The URB Handle is Dereferenced / [Opcode] == 'Dereference Resource' 1 / Keep Resource / The URBHhandle is NOT Dereferenced / [Opcode] == 'Dereference Resource'
	3:2	**Reserved** Format: MBZ
	1	**Requester Type** This field indicates whether the requesting thread is a root thread or a child thread. If it is a root thread, when Opcode is 0, FF managed resources are dereferenced. If it is a child thread and Opcode is 0, no resource is dereferenced; no action is required by the TS. Value / Name: 0 / Root Thread 1 / Child Thread
	0	**Opcode** Indicates the operation performed by the message. A root thread must terminate with a message to TS (Opcode == 0 and EOT == 1). A child thread should also terminate with such a message. A thread cannot terminate with an Opcode of "spawn thread".

Thread Spawn Message Descriptor

Value	Name	Description
0	Dereference Resource	also used for end of thread
1	Spawn Thread	

TileW SIMD8 Data Control Dword

MDCD_TILEW - TileW SIMD8 Data Control Dword

Project:	CHV, BSW
Source:	PRM
Size (in bits):	32
Default Value:	0x00000000

DWord	Bit	Description
0	31:8	**Reserved** Project: All Format: Ignore Ignored
	7:0	**Red** Project: All Format: U8 Specifies the value of the red channel to be read or written.

TileW SIMD8 Data Payload

MDP_TILEW_SIMD8 - TileW SIMD8 Data Payload

Project: CHV, BSW
Source: PRM
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0.0	31:0	**Red Slot0** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 0 red channel data
0.1	31:0	**Red Slot1** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 1 red channel data
0.2	31:0	**Red Slot2** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 2 red channel data
0.3	31:0	**Red Slot3** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 3 red channel data
0.4	31:0	**Red Slot4** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 4 red channel data
0.5	31:0	**Red Slot5** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 5 red channel data

MDP_TILEW_SIMD8 - TileW SIMD8 Data Payload		
0.6	31:0	**Red Slot6** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 6 red channel data
0.7	31:0	**Red Slot7** Project: All Format: MDCD_TileW [CHV, BSW] Specifies the Slot 7 red channel data

Transpose Message Header

MH_T - Transpose Message Header

Project: CHV, BSW
Source: DataPort 1
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:0	**X Offset** Project: All Format: S31 X offset (in bytes) of the upper left corner of the block into the surface. **Programming Notes** This field must be a multiple of the Block Width in bytes. Must be DWORD aligned.
1	31:0	**Y Offset** Project: All Format: S31 Y offset (in rows) of the upper left corner of the block into the surface. **Programming Notes** This field must be a multiple of the Block Height.
2	31:0	**Block Dimensions** Project: All Format: MHC_BDIM [CHV, BSW] The height and width of the block to transpose.
3-7	159:0	**Reserved** Project: All Format: Ignore Ignored

Untyped Write Channel Mask Message Descriptor Control Field

MDC_UW_CMASK - Untyped Write Channel Mask Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 4
Default Value: 0x00000000

DWord	Bit	Description
0	3:0	**Mask** Project: All Format: Enumeration For untyped surface write messages, indicates which channels are included in the message payload and written to the surface.

Value	Name	Description	Project
00h	RGBA **[Default]**	Red, Green, Blue, and Alpha are included	All
08h	RGB	Red, Green, and Blue are included	All
0Ch	RG	Red and Green are included	All
0Eh	R	Red is included	All
Others	Reserved	Ignored	All

Upper Oword Block Data Payload

MDP_OW1U - Upper Oword Block Data Payload		
Project:	CHV, BSW	
Source:	PRM	
Size (in bits):	256	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0.0-0.3	127:0	**Reserved** Project: All Format: Ignore Ignored
0.4-0.7	127:0	**Oword** Project: All Format: U128 Specifies the upper Oword data element

VC1

VC1		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	16	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	15:8	**Reserved** Format: MBZ
	7	**Syncmarker Error** This flag indicates missing sync marker SEs coded in the bit-stream.
	6	**Mbmode SE Error** This flag indicates inconsistent Macroblock SEs coded in the bit-stream.
	5	**Transformtype SE Error** This flag indicates inconsistent transform type SEs coded in the bit-stream.
	4	**Coefficient Error** This flag indicates inconsistent Coefficient SEs coded in the bit-stream.
	3	**Motion Vector SE Error** This flag indicates inconsistent Motion Vector SEs coded in the bit-stream.
	2	**Coded Block Pattern CY SE Error** This flag indicates inconsistent CBPCY SEs coded in the bit-stream.
	1	**Mquant Error** This flag indicates inconsistent MQUANT SEs coded in the bit-stream.
	0	**MB Concealment Flag** . Each pulse from this flag indicates one MB is concealed by hardware.

VCS Hardware-Detected Error Bit Definitions

VCS Hardware-Detected Error Bit Definitions

Project: CHV, BSW
Source: VideoCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** Project: CHV, BSW This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: Value: 1 — Name: — Description: Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

VEBOX_CAPTURE_PIPE_STATE

VEBOX_CAPTURE_PIPE_STATE

Project:	CHV, BSW
Source:	VideoEnhancementCS
Size (in bits):	96
Default Value:	0x0F12644B, 0xA064AF0A, 0xE6FD4000

This command contains variables for controlling Demosaic and the White Balance Statistics.

DWord	Bit	Description
0	31:30	**Reserved** Project: CHV, BSW Format: MBZ
	29:24	**Good Pixel Threshold** Format: U6 The difference threshold between adjacent pixels for a pixel to be considered "good". Value: Fh — Name: **[Default]** — Project: CHV, BSW
	23	**Reserved** Format: MBZ
	22:20	**Shift Min Cost** Default Value: 1h Format: U3 The amount to shift the H2/V2 versions of min_cost.
	19:16	**Scale For Average Min Cost** Default Value: 2h Project: CHV, BSW Format: U4 The amount to scale the min_cost difference during the Avg interpolation decision
	15:8	**Average Color Threshold** Format: U8 The threshold between two colors in a pixel for the Avg interpolation to be considered. Value: 64h — Name: **[Default]** — Project: CHV, BSW
	7:0	**Average Min Cost Threshold** Default Value: 4Bh Project: CHV, BSW

VEBOX_CAPTURE_PIPE_STATE

		Format: U8 The threshold for the H and V Min_cost beyond which the Avg interpolation will be used.
1	31:28	**Scale For Min Cost** Default Value: Ah The amount to scale the min_cost difference during the confidence check.
	27:24	**Reserved** Project: CHV, BSW Format: MBZ
	23:16	**Bad Color Threshold 1** Default Value: 64h Format: U8 Color value threshold used during the bad pixel check.
	15:8	**Bad Color Threshold 2** Default Value: AFh Format: U8 Color value threshold used during the bad pixel check.
	7:4	**Reserved** Project: CHV, BSW Format: MBZ
	3:0	**Bad Color Threshold 3** Default Value: Ah Format: U4 Color value threshold used during the bad pixel check.
2	31:24	**Y Bright Value** Default Value: E6h The whitepoint threshold percentile in the Y histogram. Any pixel with Y value above this could be a whitepoint. This is the larger of the calculated Ybright value and the Ythreshold value, which is the minimum Y required to be considered a white point. **Programming Notes** / **Project** "0000" is appended to the LSBs before comparing with Y. / CHV, BSW
	23:16	**Y Outlier Value** Default Value: FDh The outlier threshold percentile in the Y histogram. Any pixel with Y value above this either clipped or an outlier in the image. These points will not be included in the white patch

VEBOX_CAPTURE_PIPE_STATE

calculation.

Programming Notes	Project
"0000" is appended to the LSBs before comparing with Y.	CHV, BSW

15:8 **UV Threshold Value**

The value denotes the maximum threshold of the ratio between U+V to Y can have to be considered a gray point.

Value	Name	Description
[255,0]		Encode a value from 255/256 to 0/256
64	**[Default]**	0.25 * 255 = 64

7:0 **Reserved**

Project:	CHV, BSW
Format:	MBZ

VEBOX_Ch_Dir_Filter_Coefficient

VEBOX_Ch_Dir_Filter_Coefficient

Project: All
Source: PRM
Size (in bits): 64
Default Value: 0x00000000, 0x00000000

DWord	Bit	Description
0..1	63:56	**Filter Coefficient[7]** Format: S1.6 2's Complement **Range:** [-2, +2)
	55:48	**Filter Coefficient[6]** Format: S1.6 2's Complement **Range:** [-2, +2)
	47:40	**Filter Coefficient[5]** Format: S1.6 2's Complement **Range:** [-2, +2)
	39:32	**Filter Coefficient[4]** Format: S1.6 2's Complement **Range:** [-2, +2)
	31:24	**Filter Coefficient[3]** Format: S1.6 2's Complement **Range:** [-2, +2)
	23:16	**Filter Coefficient[2]** Format: S1.6 2's Complement **Range:** [-2, +2)
	15:8	**Filter Coefficient[1]** Format: S1.6 2's Complement **Range:** [-2, +2)
	7:0	**Filter Coefficient[0]** Format: S1.6 2's Complement **Range:** [-2, +2)

VEBOX_DNDI_STATE

VEBOX_DNDI_STATE

Project: CHV, BSW

Source: VideoEnhancementCS

Size (in bits): 320

Default Value: 0x00000800, 0x00000000, 0x04950100, 0x407D0000, 0x00000000, 0x00000000, 0x00000000, 0x105064A5, 0x00000000, 0x00000000

This state table is used by the *Denoise and Deinterlacer Functions*. When DN is used in 12-bit mode with the Capture Pipe all the DN pixel thresholds **(temporal_diff_th, temp_diff_low, good_neighbor_th)** are compared with the 8 MSBs of the 12-bit pixels.

DWord	Bit	Description
0	31:24	**Denoise STAD Threshold** Format: U8 Threshold for denoise sum of temporal absolute differences.
	23:16	**Denoise Maximum History** Format: U8 Maximum allowed value for denoise history. Value: [128,240] — Name:
	15:12	**Reserved** Format: MBZ
	11:8	**Denoise History increase** Default Value: 8h Format: U4 Amount that denoise_history is increased MAX:15
	7:0	**Denoise ASD Threshold** Format: U8 Threshold for denoise absolute sum of differences. Value: [0,63] — Name:
1	31:30	**Reserved** Format: MBZ
	29:24	**Temporal Difference Threshold** Format: U6

VEBOX_DNDI_STATE

		Programming Notes **Temporal Difference Threshold minus Low Temporal Difference Threshold** must be larger than 0 and less than or equal to 16, except when both thresholds are set to 0.
	23:22	**Reserved** Format: MBZ
	21:16	**Low Temporal Difference Threshold** Format: U6 **Programming Notes** **Temporal Difference Threshold minus Low Temporal Difference Threshold** must be larger than 0 and less than or equal to 16, except when both thresholds are set to 0.
	15:13	**STMM C2** Format: U3 Bias for divisor in STMM equation. Value / Name / Description: [0,7] / / Representing values [1,8]
	12:8	**Denoise Moving Pixel Threshold** Format: U5 Threshold for number of moving pixels to declare a block to be moving. Value / Name: [0,16] /
	7:0	**Denoise Threshold for Sum of Complexity Measure** Format: U8
2	31:30	**Reserved** Format: MBZ
	29:24	**Good Neighbor Threshold** Format: U6 Difference from current pixel for neighboring pixels to be considered a good neighbor. MAX:63 Value / Name / Description: 4 / **[Default]** / Depending on GNE of previous frame
	23:20	**Content Adaptive Threshold Slope** Format: U4 Determines the slope of the Content Adaptive Threshold. +1 added internally to get CAT_slope. Value / Name / Description: 9 / **[Default]** / CAT_slope value = 10

VEBOX_DNDI_STATE

	19:16	**SAD Tight Threshold** Default Value: 5 Format: U4
	15:14	**Smooth MV Threshold** Format: U2
	13:12	**Reserved** Format: MBZ
	11:8	**Block Noise Estimate Edge Threshold** Default Value: 1 Format: U4 Threshold for detecting an edge in block noise estimate. MAX:15
	7:0	**Block Noise Estimate Noise Threshold** Format: U8 Threshold for noise maximum/minimum. Value: [0,31] — Name:
3	31	**STMM Blending Constant Select** Format: U1 Value 0: Use the blending constant for small values of STMM for stmm_md_th Value 1: Use the blending constant for large values of STMM for stmm_md_th
	30:24	**Blending constant across time for large values of STMM** Default Value: 64 Format: U7
	23:16	**Blending constant across time for small values of STMM** Default Value: 125 Format: U8
	15:14	**Reserved** Format: MBZ
	13:8	**Multiplier for VECM** Format: U6 Determines the strength of the vertical edge complexity measure.
	7:0	**Maximum STMM** Format: U8

VEBOX_DNDI_STATE

		Largest allowed STMM in blending equations
4	31:24	**Minimum STMM** Format: U8 Smallest allowed STMM in blending equations
	23:22	**STMM Shift Down** Format: U2 Amount to shift STMM down (quantize to fewer bits) Value / Name: 0 – Shift by 4 1 – Shift by 5 2 – Shift by 6 3 – Reserved
	21:20	**STMM Shift Up** Format: U2 Amount to shift STMM up (set range). Value / Name: 0 – Shift by 6 1 – Shift by 7 2 – Shift by 8 3 – Reserved
	19:16	**STMM Output Shift** Format: U4 Amount to shift output of STMM blend equation Value / Name: [0, 16] **Programming Notes** The value of this field must satisfy the following equation: stmm_max - stmm_min = 2 ^ stmm_output_shift
	15:8	**SDI Threshold** Format: U8 Threshold for angle detection in SDI algorithm.
	7:0	**SDI Delta** Format: U8 Delta value for angle detection in SDI algorithm.

VEBOX_DNDI_STATE

5	31:24	**SDI Fallback Mode 1 T1 Constant** Format: U8
	23:16	**SDI Fallback Mode 1 T2 Constant** Format: U8
	15:8	**SDI Fallback Mode 2 Constant (Angle2x1)** Format: U8
	7:0	**FMD Temporal Difference Threshold** Format: U8
6	31:24	**FMD #1 Vertical Difference Threshold** Format: U8
	23:16	**FMD #2 Vertical Difference Threshold** Format: U8
	15:14	**CAT Threshold** Default Value: 0 Format: U2
	13:8	**FMD Tear Threshold** Format: U6
	7	**MCDI Enable** Use Motion Compensated Deinterlace algorithm. **Programming Notes** This bit is Ignored if DI Enable is off.
	6	**Progressive DN** Format: Enable Indicates that the denoise algorithm should assume progressive input when filtering neighboring pixels. **DI Enable** must be disabled when this field is enabled **Value** / **Name** 0 – DN assumes interlaced video and filters alternate lines together 1 – DN assumes progressive video and filters neighboring lines together
	5:4	**Reserved** Format: MBZ
	3	**DN/DI Top First** Format: Enable Indicates the top field is first in sequence, otherwise bottom is first **Value** / **Name** 0 – Bottom field occurs first in sequence

VEBOX_DNDI_STATE

		1 — Top field occurs first in sequence
	2:0	**Reserved** Format: MBZ
7	31:29	**Reserved** Format: MBZ
	28:23	**Initial Denoise History** Default Value: 32 Format: U6 Initial value for Denoise history for both Luma and Chroma. (Dnmh_history_init * 4) <= (Dnmh_history_max)
	22:19	**Neighbor Pixel Threshold** Default Value: 10 Format: U4
	18	**Reserved** Format: MBZ
	17:16	**Progressive Cadence Reconstruction For 2nd Field Of Previous Frame** Format: U2

Value	Name	Description
0	Deinterlace	
1	Put together with previous field in sequence	1st field of previous frame
2	Put together with next field in sequence	1st field of current frame

	15:10	**MC Pixel Consistency Threshold** Default Value: 25 Format: U6
	9:8	**Progressive Cadence Reconstruction for 1st Field of Current Frame** Format: U2

Value	Name	Description
0	Deinterlace	
1	Put together with previous field in sequence	2nd field of previous frame
2	Put together with next field in sequence	2nd field of current frame

	7:4	**SAD THB** Default Value: 10 Format: U4

VEBOX_DNDI_STATE

	3:0	**SAD THA** Default Value: 5 Format: U4
8	31:24	**Reserved** Format: MBZ
	23:16	**Chroma Denoise STAD Threshold** Format: U8 Threshold for denoise sum of temporal absolute differences.
	15:13	**Reserved** Format: MBZ
	12	**Chroma Denoise Enable** Value 1: The U and V chroma channels will be denoise filtered. Value 0: The U and V channels will be passed to the next stage after DN unchanged.
	11:6	**Chroma Temporal Difference Threshold** Format: U6 **Programming Notes** 0<**[Chroma Temporal Difference Threshold - Chroma Low Temporal Difference Threshold]**«16 (Larger than 0 and less than or equal to 16)
	5:0	**Chroma Low Temporal Difference Threshold** Format: U6 **Programming Notes** 0<**[Chroma Temporal Difference Threshold - Chroma Low Temporal Difference Threshold]**«16 (Larger than 0 and less than or equal to 16)
9	31:12	**Reserved** Format: MBZ
	11:8	**Hot Pixel Count** Format: U4 Number of neighboring pixels different more than **HotPixThr** before a pixel is considered hot. Value: [0,8] **Programming Notes** 0 will cause all pixels to be considered hot and will perform a median filter on the entire image.

VEBOX_DNDI_STATE		
	7:0	**Hot Pixel Threshold** Format: U8 Threshold for a difference from the value of a neighboring pixel. Is shifted up to 12-bits before compare.

VEBOX_Filter_Coefficient

VEBOX_Filter_Coefficient		
Project:	All	
Source:	PRM	
Size (in bits):	8	
Default Value:	0x00000000	
DWord	**Bit**	**Description**
0	7:0	**2's Complement Filter Coefficient** Format: S1.6 2's Complement **Range:** [-2, +2)

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

Project: CHV, BSW

Source: VideoEnhancementCS

Size (in bits): 384

Default Value: 0x4F371E00, 0xA28D7A65, 0xEDDBC8B5, 0x21140A03, 0x755C4331, 0x00D7B493, 0x0048001A, 0x0097006B, 0x00F300C3, 0x01510131, 0x01BD0194, 0x022B01F2

This state structure contains the Forward Gamma Correction state.

DWord	Bit	Description
0	31:24	**PWL_Fwd_Gamma_Point 3** Default Value: 79 Format: U8
	23:16	**PWL_Fwd_Gamma_Point 2** Default Value: 55 Format: U8
	15:8	**PWL_Fwd_Gamma_Point 1** Default Value: 30 Format: U8
	7:1	**Reserved** Format: MBZ
	0	**Forward Gamma Correction Enable** Format: Enable **Programming Notes** / **Project** **Demosaic** must also be enabled if this is enabled. / CHV, BSW
1	31:24	**PWL_Fwd_Gamma_Point 7** Default Value: 162 Format: U8
	23:16	**PWL_Fwd_Gamma_Point 6** Default Value: 141 Format: U8
	15:8	**PWL_Fwd_Gamma_Point 5** Default Value: 122 Format: U8
	7:0	**PWL_Fwd_Gamma_Point 4** Default Value: 101

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

		Format:	U8
2	31:24	**PWL_Fwd_Gamma_Point 11**	
		Default Value:	237
		Format:	U8
	23:16	**PWL_Fwd_Gamma_Point 10**	
		Default Value:	219
		Format:	U8
	15:8	**PWL_Fwd_Gamma_Point 9**	
		Default Value:	200
		Format:	U8
	7:0	**PWL_Fwd_Gamma_Point 8**	
		Default Value:	181
		Format:	U8
3	31:24	**PWL_Fwd_Gamma_Bias_4**	
		Default Value:	33
		Format:	U8
	23:16	**PWL_Fwd_Gamma_Bias_3**	
		Default Value:	20
		Format:	U8
	15:8	**PWL_Fwd_Gamma_Bias_2**	
		Default Value:	10
		Format:	U8
	7:0	**PWL_Fwd_Gamma_Bias_1**	
		Default Value:	3
		Format:	U8
4	31:24	**PWL_Fwd_Gamma_Bias_8**	
		Default Value:	117
		Format:	U8
	23:16	**PWL_Fwd_Gamma_Bias_7**	
		Default Value:	92
		Format:	U8
	15:8	**PWL_Fwd_Gamma_Bias_6**	
		Default Value:	67
		Format:	U8
	7:0	**PWL_Fwd_Gamma_Bias_5**	

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

		Default Value: 49 Format: U8
5	31:24	**Reserved** Format: MBZ
	23:16	**PWL_Fwd_Gamma_Bias_11** Default Value: 215 Format: U8
	15:8	**PWL_Fwd_Gamma_Bias_10** Default Value: 180 Format: U8
	7:0	**PWL_Fwd_Gamma_Bias_9** Default Value: 147 Format: U8
6	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Fwd_Gamma_Slope_1** Default Value: 048h 72/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_0** Default Value: 01Ah 26/256 Format: U4.8
7	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Fwd_Gamma_Slope_3** Default Value: 097h 151/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_2** Default Value: 06Bh 107/256 Format: U4.8
8	31:28	**Reserved** Format: MBZ

VEBOX_FORWARD_GAMMA_CORRECTION_STATE

	27:16	**PWL_Fwd_Gamma_Slope_5** Default Value: 0F3h 243/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_4** Default Value: 0C3h 195/256 Format: U4.8
9	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Fwd_Gamma_Slope_7** Default Value: 151h 337/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_6** Default Value: 131h 305/256 Format: U4.8
10	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Fwd_Gamma_Slope_9** Default Value: 1BDh 445/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_8** Default Value: 194h 404/256 Format: U4.8
11	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Fwd_Gamma_Slope_11** Default Value: 22Bh 555/256 Format: U4.8
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Fwd_Gamma_Slope_10**

VEBOX_FORWARD_GAMMA_CORRECTION_STATE			
		Default Value:	1F2h 498/256
		Format:	U4.8

VEBOX_GAMUT_STATE

VEBOX_GAMUT_STATE

Project: CHV, BSW

Source: VideoEnhancementCS

Size (in bits): 1216

Default Value: 0x01B40000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x09050201, 0x412A1A10, 0x00BB8860, 0x3526170D, 0x8B725B47, 0x00DFC1A5, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x654F371E, 0x00000000, 0x00EDDBC8, 0x21140A03, 0x755C4331, 0x00D7B493, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x0CD2911F, 0xB0000334, 0x00000000

DWord	Bit	Description
0	31:25	**Reserved** Format: MBZ
	24:16	**A(r)** Default Value: 436 Format: U9 Gain_factor_R (default: 436, preferred range: 256-511)
	15	**Global Mode Enable** The gain factor derived from state CM(w) Value 0: Advance Mode Value 1: Basic Mode
	14:10	**Reserved** Format: MBZ
	9:0	**CM(w)** Format: U10 WeightingFactorForGain_factor (only enabled when the GlobalModeEnable is on)
1	31:26	**Reserved** Format: MBZ
	25:16	**CM(s)** Format: U2.8 AccurateColorComponentScaling (default: 640/256, preferred range: [512-1023]/256) The default is 640/256

VEBOX_GAMUT_STATE

	15	**Reserved** Format: MBZ
	14:8	**A(g)** Format: U7 Gain_factor_G (default: 26/256, preferred range: [26-127]/256) The default is 26/256
	7	**Reserved** Format: MBZ
	6:0	**A(b)** Format: U7 Gain_factor_B (default: 26/256, preferred range: [26-127]/256) The default is 26/256
2	31:26	**Reserved** Format: MBZ
	25:16	**R(s)** Format: U2.8 RedScaling (default: 768/256, preferred range: [512-1023]/256) The default is 768/256
	15:8	**CM(i)** Format: U0.8 AccurateColorComponentOffset (default: 192/256, preferred range: [0-192]/256) The default is 192/256
	7:0	**R(i)** Format: U0.8 RedOffset (default: 128/256, preferred range: [0-128]/256) The default is 128/256
3	31	**Reserved** Format: MBZ
	30:16	**C1** Format: S2.12 Coefficient of 3x3 Transform matrix

VEBOX_GAMUT_STATE

		The default is 1141/4096
	15	**Reserved** Format: MBZ
	14:0	**C0** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 2792/4096
4	31	**Reserved** Format: MBZ
	30:16	**C3** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 71/4096
	15	**Reserved** Format: MBZ
	14:0	**C2** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 34/4096
5	31	**Reserved** Format: MBZ
	30:16	**C5** Format: S2.12 Coefficient of 3x3 Transform matrix The default is -52/4096
	15	**Reserved** Format: MBZ
	14:0	**C4** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 3663/4096
6	31	**Reserved**

VEBOX_GAMUT_STATE

		Format: MBZ
	30:16	**C7** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 168/4096
	15	**Reserved** Format: MBZ
	14:0	**C6** Format: S2.12 Coefficient of 3x3 Transform matrix The default is -12/4096
7	31:15	**Reserved** Format: MBZ
	14:0	**C8** Format: S2.12 Coefficient of 3x3 Transform matrix The default is 3434/4096
8	31:24	**PWL_Gamma_ Point 4** Default Value: 9 Format: U8 Point 4 for PWL for gamma correction
	23:16	**PWL_Gamma_ Point 3** Default Value: 5 Format: U8 Point 3 for PWL for gamma correction
	15:8	**PWL_Gamma_ Point 2** Default Value: 2 Format: U8 Point 2 for PWL for gamma correction
	7:0	**PWL_Gamma_ Point 1** Default Value: 1 Format: U8

VEBOX_GAMUT_STATE

		Point 1 for PWL for gamma correction
9	31:24	**PWL_Gamma_ Point 8** Default Value: 65 Point 8 for PWL for gamma correction
	23:16	**PWL_Gamma_ Point 7** Default Value: 42 Point 7 for PWL for gamma correction
	15:8	**PWL_Gamma_ Point 6** Default Value: 26 Point 6 for PWL for gamma correction
	7:0	**PWL_Gamma_ Point 5** Default Value: 16 Point 5 for PWL for gamma correction
10	31:24	**Reserved** Format: MBZ
	23:16	**PWL_Gamma_ Point 11** Default Value: 187 Format: U8 Point 11 for PWL for gamma correction
	15:8	**PWL_Gamma_ Point 10** Default Value: 136 Format: U8 Point 10 for PWL for gamma correction
	7:0	**PWL_Gamma_ Point 9** Default Value: 96 Format: U8 Point 9 for PWL for gamma correction
11	31:24	**PWL_Gamma_ Bias_4** Default Value: 53 Format: U8 Bias 4 for PWL for gamma correction

VEBOX_GAMUT_STATE

	23:16	**PWL_Gamma_ Bias_3** Default Value: 38 Format: U8 Bias 3 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_2** Default Value: 23 Format: U8 Bias 2 for PWL for gamma correction
	7:0	**PWL_Gamma_ Bias_1** Default Value: 13 Format: U8 Bias 1 for PWL for gamma correction
12	31:24	**PWL_Gamma_ Bias_8** Default Value: 139 Format: U8 Bias 8 for PWL for gamma correction
	23:16	**PWL_Gamma_ Bias_7** Default Value: 114 Format: U8 Bias 7 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_6** Default Value: 91 Format: U8 Bias 6 for PWL for gamma correction
	7:0	**PWL_Gamma_ Bias_5** Default Value: 71 Format: U8 Bias 5 for PWL for gamma correction
13	31:24	**Reserved** Format: MBZ
	23:16	**PWL_Gamma_ Bias_11**

VEBOX_GAMUT_STATE

		Default Value: 223 Format: U8 Bias 11 for PWL for gamma correction
	15:8	**PWL_Gamma_ Bias_10** Default Value: 193 Format: U8 Bias 10 for PWL for gamma correction
	7:0	**PWL_Gamma_ Bias_9** Default Value: 165 Format: U8 Bias 9 for PWL for gamma correction
14	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_1** Format: U4.8 Slope 1 for PWL for gamma correction The default is 2560/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_0** Format: U4.8 Slope 0 for PWL for gamma correction The default is 3328/256
15	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_3** Format: U4.8 Slope 3 for PWL for gamma correction The default is 960/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_2**

VEBOX_GAMUT_STATE

		Format: U4.8 Slope 2 for PWL for gamma correction The default is 1280/256
16	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_5** Format: U4.8 Slope 5 for PWL for gamma correction The default is 512/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_4** Format: U4.8 Slope 4 for PWL for gamma correction The default is 658/256
17	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_7** Format: U4.8 Slope 7 for PWL for gamma correction The default is 278/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_6** Format: U4.8 Slope 6 for PWL for gamma correction The default is 368/256
18	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_9** Format: U4.8

VEBOX_GAMUT_STATE

		Slope 9 for PWL for gamma correction The default is 179/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_8** Format: U4.8 Slope 8 for PWL for gamma correction The default is 215/256
19	31:28	**Reserved** Format: MBZ
	27:16	**PWL_Gamma_ Slope_11** Format: U4.8 Slope 11 for PWL for gamma correction The default is 124/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_Gamma_ Slope_10** Format: U4.8 Slope 10 for PWL for gamma correction The default is 151/256
20	31:24	**PWL_INV_GAMMA_ Point 4** Default Value: 101 Format: U8 Point 4 for PWL for inverse gamma correction
	23:16	**PWL_INV_GAMMA_ Point 3** Default Value: 79 Format: U8 Point 3 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Point 2** Default Value: 55 Format: U8 Point 2 for PWL for inverse gamma correction

VEBOX_GAMUT_STATE

	7:0	**PWL_INV_GAMMA_ Point 1** Default Value: 30 Format: U8 Point 1 for PWL for inverse gamma correction
21	31:24	**PWL_INV_GAMMA_ Point 8** Format: U8 Point 8 for PWL for inverse gamma correction Value: 181
	23:16	**PWL_INV_GAMMA_ Point 7** Format: U8 Point 7 for PWL for inverse gamma correction Value: 162
	15:8	**PWL_INV_GAMMA_ Point 6** Format: U8 Point 6 for PWL for inverse gamma correction Value: 141
	7:0	**PWL_INV_GAMMA_ Point 5** Format: U8 Point 5 for PWL for inverse gamma correction Value: 122
22	31:24	**Reserved** Format: MBZ
	23:16	**PWL_INV_GAMMA_ Point 11** Default Value: 237 Format: U8 Point 11 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Point 10** Default Value: 219 Format: U8 Point 10 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Point 9**

VEBOX_GAMUT_STATE

		Default Value: 200 Format: U8 Point 9 for PWL for inverse gamma correction
23	31:24	**PWL_INV_GAMMA_ Bias_4** Default Value: 33 Format: U8 Bias 4 for PWL for inverse gamma correction
	23:16	**PWL_INV_GAMMA_ Bias_3** Default Value: 20 Format: U8 Bias 3 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_2** Default Value: 10 Format: U8 Bias 2 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Bias_1** Default Value: 3 Format: U8 Bias 1 for PWL for inverse gamma correction
24	31:24	**PWL_INV_GAMMA_ Bias_8** Default Value: 117 Format: U8 Bias 8 for PWL for inverse gamma correction
	23:16	**PWL_INV_GAMMA_ Bias_7** Default Value: 92 Format: U8 Bias 7 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_6** Default Value: 67 Format: U8 Bias 6 for PWL for inverse gamma correction

VEBOX_GAMUT_STATE

	7:0	**PWL_INV_GAMMA_ Bias_5** Default Value: 49 Format: U8 Bias 5 for PWL for inverse gamma correction
25	31:24	**Reserved** Format: MBZ
	23:16	**PWL_INV_GAMMA_ Bias_11** Default Value: 215 Format: U8 Bias 11 for PWL for inverse gamma correction
	15:8	**PWL_INV_GAMMA_ Bias_10** Default Value: 180 Format: U8 Bias 10 for PWL for inverse gamma correction
	7:0	**PWL_INV_GAMMA_ Bias_9** Default Value: 147 Format: U8 Bias 9 for PWL for inverse gamma correction
26	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_1** Format: U4.8 Slope 1 for PWL for gamma correction The default is 72/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_0** Format: U4.8 Slope 0 for PWL for gamma correction The default is 26/256
27	31:28	**Reserved** Format: MBZ

VEBOX_GAMUT_STATE		
	27:16	**PWL_INV_GAMMA_ Slope_3** Format: U4.8 Slope 3 for PWL for gamma correction The default is 151/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_2** Format: U4.8 Slope 2 for PWL for gamma correction The default is 107/256
28	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_5** Format: U4.8 Slope 5 for PWL for gamma correction The default is 243/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_4** Format: U4.8 Slope 4 for PWL for gamma correction The default is 195/256
29	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_7** Format: U4.8 Slope 7 for PWL for gamma correction The default is 337/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_6** Format: U4.8

VEBOX_GAMUT_STATE

		Slope 6 for PWL for gamma correction The default is 305/256
30	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_9** Format: U4.8 Slope 9 for PWL for gamma correction The default is 445/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_8** Format: U4.8 Slope 8 for PWL for gamma correction The default is 404/256
31	31:28	**Reserved** Format: MBZ
	27:16	**PWL_INV_GAMMA_ Slope_11** Format: U4.8 Slope 11 for PWL for gamma correction The default is 555/256
	15:12	**Reserved** Format: MBZ
	11:0	**PWL_INV_GAMMA_ Slope_10** Format: U4.8 Slope 10 for PWL for gamma correction The default is 498/256
32	31	**Reserved** Format: MBZ
	30:16	**Offset_in_G** Default Value: 0 Format: S14 The input offset for green component

VEBOX_GAMUT_STATE

	15	**Reserved** Format: MBZ
	14:0	**Offset_in_R** Default Value: 0 Format: S14 The input offset for red component
33	31	**Reserved** Format: MBZ
	30:16	**Offset_out_B** Format: S2.12 The input offset for green component The default is -1246/4096
	15	**Reserved** Format: MBZ
	14:0	**Offset_in_B** Default Value: 0 Format: S14 The input offset for red component
34	31	**Reserved** Format: MBZ
	30:16	**Offset_out_G** Format: S2.12 The input offset for green component The default is -983/4096
	15	**Reserved** Format: MBZ
	14:0	**Offset_out_R** Format: S2.12 The input offset for red component The default is -974/4096
35	31	**Reserved**

VEBOX_GAMUT_STATE

		Format: MBZ
	30	**FullRangeMappingEnable** Value: 0 — Basic Mode **[Default]** Value: 1 — Advance Mode
	29:20	**d(in,default)** Default Value: 205 Format: U10 InnerTriangleMappingLength
	19:10	**d(out, default)** Default Value: 164 Format: U10 OuterTriangleMappingLength
	9:0	**d1(out)** Default Value: 287 Format: U10 OuterTriangleMappingLengthBelow
36	31	**xvYccDecEncEnable** This bit is valid only when ColorGamutCompressionnEnable is on. Value: 1 — Both xvYcc decode and xvYcc encode are enabled **[Default]** Value: 0 — To disable both xvYcc decode and xvYcc encode
	30:28	**CompressionLineShift** Value: 3 — **[Default]** Value: [0,4]
	27:10	**Reserved** Format: MBZ
	9:0	**d1(in)** Default Value: 820 Format: U10 InnerTriangleMappingLengthBelow
37	31:30	**GCC BasicModeSelection**

VEBOX_GAMUT_STATE

		Value / **Name** / **Description** 00b / Default / 01b / Scaling Factor / Used along with Dword66 Bits 28:11 10b / Single Axis Gamma Correction / Used along with Dword67 Bit 29 11b / Scaling factor with fixed luma / Used along with Dword37 Bits 28:11
	29	**LumaChormaOnlyCorrection** **Value** / **Name** 0 / Luma Only Correction **[Default]** 1 / Chorma Only Correction
	28:25	**Reserved** Project: CHV, BSW Format: MBZ
	24:11	**BasicModeScalingFactor** Project: CHV, BSW Format: U2.12 Used when FullRangeMappingEnable is in basic mode and base mode selection bit is set to scaling factor.
	10:1	**Reserved** Format: MBZ
	0	**Cpi Override** **Value** / **Name** 0 / **[Default]** 1 / Override Cpi calculation

VEBOX_RGB_TO_GAMMA_CORRECTION

VEBOX_RGB_TO_GAMMA_CORRECTION		
Source:	VideoEnhancementCS	
Size (in bits):	64	
Default Value:	0x00000000, 0x00000000	
Color depth is 16 bits.		
DWord	**Bit**	**Description**
0..1	63:48	**B-ch Corrected Value** Default Value: 0h Format: U16
	47:32	**G-ch Corrected Value** Default Value: 0h Format: U16
	31:16	**R-ch Corrected Value** Default Value: 0h Format: U16
	15:0	**Pixel Value** Default Value: 0h Format: U16

VEBOX_STD_STE_STATE

VEBOX_STD_STE_STATE

Project:	CHV, BSW
Source:	VideoEnhancementCS
Size (in bits):	928
Default Value:	0x9A6E39F0, 0x400D3C65, 0x000C9180, 0xFE2F2E00, 0x0003FFFF, 0x00140000, 0xD82E0640, 0x8285ECEC, 0x07FB8282, 0x00000000, 0x02117000, 0xA38FEC96, 0x0100C8C8, 0x003A6871, 0x01478000, 0x0107C306, 0x1291F008, 0x00094855, 0x1C1BD100, 0x03802008, 0x0002A980, 0x00080180, 0x0007CFF5, 0x18D1F07C, 0x000800BD, 0x1C080100, 0x03800000, 0x0008012B, 0x0008012B

This state structure contains the state used by the STD/STE function.

DWord	Bit	Description
0	31:24	**V_Mid** Default Value: 154 Format: U8 Rectangle middle-point V coordinate.
	23:16	**U_Mid** Default Value: 110 Format: U8 Rectangle middle-point U coordinate.
	15:10	**Hue_Max** Default Value: 14 Format: U6 Rectangle half width.
	9:4	**Sat_Max** Default Value: 31 Format: U6 Rectangle half length.
	3	**Reserved** Format: MBZ
	2	**Output Control** Value 0: Output Pixels Value 1: Output STD Decisions

VEBOX_STD_STE_STATE

	1	**STE Enable** Format: Enable
	0	**STD Enable** Format: Enable **Programming Notes** This needs to be enabled if 'STD Score Output' is enabled.
1	31	**Reserved** Project: CHV, BSW Format: MBZ
	30:28	**Diamond Margin** Default Value: 4 Format: U3
	27:21	**Diamond_du** Default Value: 0 Format: S6 2's complement Rhombus center shift in the sat-direction, relative to the rectangle center.
	20:18	**HS_margin** Default Value: 3 Format: U3 Defines rectangle margin.
	17:10	**Cos(α)** Default Value: 79 Format: S0.7 2's complement The default is 79/128
	9:8	**Reserved** Format: MBZ
	7:0	**Sin(α)** Default Value: 101 Format: S0.7 2's complement The default is 101/128
2	31:21	**Reserved** Format: MBZ
	20:13	**Diamond_alpha**

VEBOX_STD_STE_STATE

		Default Value: 100 Format: U2.6 1/tan(β) The default is 100/64
	12:7	**Diamond_Th** Default Value: 35 Format: U6 Half length of the rhombus axis in the sat-direction.
	6:0	**Diamond_dv** Default Value: 0 Format: S6 2's complement Rhombus center shift in the hue-direction, relative to the rectangle center.
3	31:24	**Y_point_3** Default Value: 254 Format: U8 Third point of the Y piecewise linear membership function.
	23:16	**Y_point_2** Default Value: 47 Format: U8 Second point of the Y piecewise linear membership function.
	15:8	**Y_point_1** Default Value: 46 Format: U8 First point of the Y piecewise linear membership function.
	7	**VY_STD_Enable** Format: Enable Enables STD in the VY subspace.
	6:0	**Reserved** Format: MBZ
4	31:18	**Reserved** Format: MBZ
	17:13	**Y_Slope_2** Default Value: 31

VEBOX_STD_STE_STATE

		Format: U2.3 Slope between points Y3 and Y4. The default is 31/8
	12:8	**Y_Slope_1** Default Value: 31 Format: U2.3 Slope between points Y1 and Y2. The default is 31/8
	7:0	**Y_point_4** Default Value: 255 Format: U8 Fourth point of the Y piecewise linear membership function.
5	31:16	**INV_Skin_types_margin** Default Value: 20 Skin_Type_margin Format: U0.16 1/(2* Skin_types_margin)
	15:0	**INV_Margin_VYL** Format: U0.16 1 / Margin_VYL 1/ Margin_VYL = 3300/65536
6	31:24	**P1L** Default Value: 216 Format: U8 Y Point 1 of the lower part of the detection PWLF.
	23:16	**P0L** Default Value: 46 Format: U8 Y Point 0 of the lower part of the detection PWLF.
	15:0	**INV_Margin_VYU** Default Value: 1600 Format: U0.16 1 / Margin_VYU = 1600/65536

VEBOX_STD_STE_STATE

7	31:24	**B1L** Default Value: 130 Format: U8 V Bias 1 of the lower part of the detection PWLF.
	23:16	**B0L** Default Value: 133 Format: U8 V Bias 0 of the lower part of the detection PWLF.
	15:8	**P3L** Default Value: 236 Format: U8 Y Point 3 of the lower part of the detection PWLF.
	7:0	**P2L** Default Value: 236 Format: U8 Y Point 2 of the lower part of the detection PWLF.
8	31:27	**Reserved** Format: MBZ
	26:16	**S0L** Default Value: FFBh Format: S2.8 2's complement Slope 0 of the lower part of the detection PWLF. The default is -5/256
	15:8	**B3L** Default Value: 130 Format: U8 V Bias 3 of the lower part of the detection PWLF.
	7:0	**B2L** Default Value: 130 Format: U8 V Bias 2 of the lower part of the detection PWLF.
9	31:22	**Reserved**

VEBOX_STD_STE_STATE

		Format: MBZ
	21:11	**S2L** Default Value: 0 Format: S2.8 2's complement The default is 0/256
	10:0	**S1L** Default Value: 0 Format: S2.8 2's complement Slope 1 of the lower part of the detection PWLF. The default is 0/256
10	31:27	**Reserved** Format: MBZ
	26:19	**P1U** Default Value: 66 Format: U8 Y Point 1 of the upper part of the detection PWLF.
	18:11	**P0U** Default Value: 46 Format: U8 Y Point 0 of the upper part of the detection PWLF.
	10:0	**S3L** Default Value: 0 Format: S2.8 2's complement Slope 3 of the lower part of the detection PWLF. The default is 0/256
11	31:24	**B1U** Default Value: 163 Format: U8 V Bias 1 of the upper part of the detection PWLF.
	23:16	**B0U** Default Value: 143 Format: U8

VEBOX_STD_STE_STATE

		V Bias 0 of the upper part of the detection PWLF.
	15:8	**P3U** Default Value: 236 Format: U8 Y Point 3 of the upper part of the detection PWLF.
	7:0	**P2U** Default Value: 150 Format: U8 Y Point 2 of the upper part of the detection PWLF.
12	31:27	**Reserved** Format: MBZ
	26:16	**S0U** Default Value: 256 Format: S2.8 2's complement Slope 0 of the upper part of the detection PWLF. The default is 256/256
	15:8	**B3U** Default Value: 200 Format: U8 V Bias 3 of the upper part of the detection PWLF.
	7:0	**B2U** Default Value: 200 Format: U8 V Bias 2 of the upper part of the detection PWLF.
13	31:22	**Reserved** Format: MBZ
	21:11	**S2U** Default Value: F4Dh Format: S2.8 2's complement Slope 2 of the upper part of the detection PWLF. The default is -179/256

VEBOX_STD_STE_STATE

	10:0	**S1U** Default Value: 113 Format: S2.8 Slope 1 of the upper part of the detection PWLF. The default is 113/256
14	31:28	**Reserved** Format: MBZ
	27:20	**Skin_types_margin** Default Value: 20 Format: U8 Skin types Y margin Restrict Skin_types_thresh >= Skin_types_margin > 0 Restrict (Skin_types_thresh + Skin_types_margin) <= 255
	19:12	**Skin_types_thresh** Default Value: 120 Format: U8 Skin types Y margin Restrict Skin_types_thresh >= Skin_types_margin > 0 Restrict (Skin_types_thresh + Skin_types_margin) <= 255
	11	**Skin_Types_Enable** Default Value: 0 Disable Format: Enable Treat differently bright and dark skin types
	10:0	**S3U** Default Value: 0 Format: S2.8 2's complement Slope 3 of the upper part of the detection PWLF. The default is 0/256
15	31	**Reserved** Format: MBZ
	30:21	**SATB1** Default Value: 8 Format: S7.2 2's complement First bias for the saturation PWLF (bright skin).

VEBOX_STD_STE_STATE

		The default is 8/4
	20:14	**SATP3** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (bright skin).
	13:7	**SATP2** Default Value: 6 Format: S6 2's complement Second point for the saturation PWLF (bright skin).
	6:0	**SATP1** Default Value: 6 Format: S6 2's complement First point for the saturation PWLF (bright skin).
16	31	**Reserved** Format: MBZ
	30:20	**SATS0** Default Value: 297 Format: U3.8 Zeroth slope for the saturation PWLF (bright skin) The default is 297/256
	19:10	**SATB3** Default Value: 124 Format: S7.2 2's complement Third bias for the saturation PWLF (bright skin) The default is 124/4
	9:0	**SATB2** Default Value: 8 Format: S7.2 2's complement Second bias for the saturation PWLF (bright skin) The default is 8/4
17	31:22	**Reserved** Format: MBZ

VEBOX_STD_STE_STATE

	21:11	**SATS2** Default Value: 297 Format: U3.8 Second slope for the saturation PWLF (bright skin) The default is 297/256
	10:0	**SATS1** Default Value: 85 Format: U3.8 First slope for the saturation PWLF (bright skin) The default is 85/256
18	31:25	**HUEP3** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (bright skin)
	24:18	**HUEP2** Default Value: 6 Format: S6 2's complement Second point for the hue PWLF (bright skin)
	17:11	**HUEP1** Default Value: 7Ah -6 Format: S6 2's complement First point for the hue PWLF (bright skin)
	10:0	**SATS3** Default Value: 256 Format: U3.8 Third slope for the saturation PWLF (bright skin) The default is 256/256
19	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3** Default Value: 56 Format: S7.2 2's complement

VEBOX_STD_STE_STATE

		Third bias for the hue PWLF (bright skin) The default is 56/4
	19:10	**HUEB2** Default Value: 8 Format: S7.2 2's complement Second bias for the hue PWLF (bright skin) The default is 8/4
	9:0	**HUEB1** Default Value: 8 Format: S7.2 2's complement First bias for the hue PWLF (bright skin) The default is 8/4
20	31:22	**Reserved** Format: MBZ
	21:11	**HUES1** Default Value: 85 Format: U3.8 First slope for the hue PWLF (bright skin) The default is 85/256
	10:0	**HUES0** Default Value: 384 Format: U3.8 Zeroth slope for the hue PWLF (bright skin) The default is 384/256
21	31:22	**Reserved** Format: MBZ
	21:11	**HUES3** Default Value: 256 Format: U3.8 Third slope for the hue PWLF (bright skin) The default is 256/256

VEBOX_STD_STE_STATE

	10:0	**HUES2** Default Value: 384 Format: U3.8 Second slope for the hue PWLF (bright skin) The default is 384/256
22	31	**Reserved** Format: MBZ
	30:21	**SATB1_DARK** Default Value: 0 Format: S7.2 2's complement First bias for the saturation PWLF (dark skin) The default is 0/4
	20:14	**SATP3_DARK** Default Value: 31 Format: S6 2's complement Third point for the saturation PWLF (dark skin)
	13:7	**SATP2_DARK** Default Value: 31 Format: S6 2's complement Second point for the saturation PWLF (dark skin)
	6:0	**SATP1_DARK** Default Value: FF5h Format: S6 2's complement First point for the saturation PWLF (dark skin) Default Value: -11
23	31	**Reserved** Format: MBZ
	30:20	**SATS0_DARK** Default Value: 397 Format: U3.8 Zeroth slope for the saturation PWLF (dark skin) The default is 397/256
	19:10	**SATB3_DARK**

VEBOX_STD_STE_STATE

		Default Value: 124 Format: S7.2 2's complement Third bias for the saturation PWLF (dark skin) The default is 124/4
	9:0	**SATB2_DARK** Default Value: 124 Format: S7.2 2's complement Second bias for the saturation PWLF (dark skin) The default is 124/4
24	31:22	**Reserved** Format: MBZ
	21:11	**SATS2_DARK** Default Value: 256 Format: U3.8 Second slope for the saturation PWLF (dark skin) The default is 256/256
	10:0	**SATS1_DARK** Default Value: 189 Format: U3.8 First slope for the saturation PWLF (dark skin) The default is 189/256
25	31:25	**HUEP3_DARK** Default Value: 14 Format: S6 2's complement Third point for the hue PWLF (dark skin).
	24:18	**HUEP2_DARK** Default Value: 2 Format: S6 2's complement Second point for the hue PWLF (dark skin).
	17:11	**HUEP1_DARK** Default Value: 0 Format: S6 2's complement

VEBOX_STD_STE_STATE

		First point for the hue PWLF (dark skin).
	10:0	**SATS3_DARK** Default Value: 256 Format: U3.8 Third slope for the saturation PWLF (dark skin) The default is 256/256
26	31:30	**Reserved** Format: MBZ
	29:20	**HUEB3_DARK** Default Value: 56 Format: S7.2 2's complement Third bias for the hue PWLF (dark skin). The default is 56/4
	19:10	**HUEB2_DARK** Default Value: 0 Format: S7.2 2's complement Second bias for the hue PWLF (dark skin). The default is 0/4
	9:0	**HUEB1_DARK** Default Value: 0 Format: S7.2 2's complement First bias for the hue PWLF (dark skin). The default is 0/4
27	31:22	**Reserved** Format: MBZ
	21:11	**HUES1_DARK** Default Value: 256 Format: U3.8 First slope for the hue PWLF (dark skin). The default is 256/256
	10:0	**HUES0_DARK**

VEBOX_STD_STE_STATE

		Default Value: 299 Format: U3.8 Zeroth slope for the hue PWLF (dark skin). The default is 299/256
28	31:22	**Reserved** Format: MBZ
	21:11	**HUES3_DARK** Default Value: 256 Format: U3.8 Third slope for the hue PWLF (dark skin). The default is 256/256
	10:0	**HUES2_DARK** Default Value: 299 Format: U3.8 Second slope for the hue PWLF (dark skin). The default is 299/256

VEBOX TLB Control Register

VTCR - VEBOX TLB Control Register

Register Space:	MMIO: 0/2/0
Project:	CHV, BSW
Source:	PRM
Default Value:	0x00000000
Size (in bits):	32
Address:	04270h

DWord	Bit	Description
0	31:1	**Reserved** Default Value: 00000000000000000000000000000000b Access: RO
	0	**Invalidate TLBs on the corresponding Engine** Default Value: 0b Access: R/W SW writes 1 to invalidate the TLBs for the associated engine and HW clears the bit when invalidation is complete. To ensure proper invalidation of the TLBs, SW has to ensure the corresponding engine's HW pipeline is flushed and cleared from all its memory accesses. Otherwise HW cannot guarantee the proper invalidation for TLBs. This bit is self clear.

VEBOX_VERTEX_TABLE

VEBOX_VERTEX_TABLE	
Project:	CHV, BSW
Source:	VideoEnhancementCS
Size (in bits):	16384
Default Value:	0x00000000, 0x00000000,

VEBOX_VERTEX_TABLE
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,
0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000,

VEBOX_VERTEX_TABLE

0x00000000, 0x00000000

DWord	Bit	Description
0..511	31:28	**Reserved** Format: MBZ
	27:16	**Vertex table entry 0 - Lv (12 bits)** Value: 100h-ED6h; Name: ; Description: Range for Vertices BT601 and BT709
	15:12	**Reserved** Format: MBZ
	11:0	**Vertex table entry 0 - Cv (12 bits)** Value: 400h-A00h; Name: ; Description: Range for Vertices BT601 and BT709

VECS Hardware-Detected Error Bit Definitions

VECS Hardware-Detected Error Bit Definitions

Project: CHV, BSW
Source: VideoEnhancementCS
Size (in bits): 16
Default Value: 0x00000000

DWord	Bit	Description
0	15:3	**Reserved** Format: MBZ
	2	**Command Privilege Violation Error** Project: CHV, BSW This bit is set if a command classified as privileged is parsed in a non-privileged batch buffer. The command will be converted to a NOOP and parsing will continue.
	1	**Reserved** Format: MBZ
	0	**Instruction Error** This bit is set when the Renderer Instruction Parser detects an error while parsing an instruction. Instruction errors include: • Client ID value (Bits 31:29 of the Header) is not supported (only MI, 2D and 3D are supported). • Defeatured MI Instruction Opcodes: Value: 1 — Name: — Description: Instruction Error detected **Programming Notes** This error indications cannot be cleared except by reset (i.e., it is a fatal error).

VERTEX_BUFFER_STATE

VERTEX_BUFFER_STATE

Project: CHV, BSW
Source: RenderCS
Size (in bits): 128
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000

This structure is used in 3DSTATE_VERTEX_BUFFERS to set the state associated with a VB. The VF function will use this state to determine how/where to extract vertex element data for all vertex elements associated with the VB.

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Project: All Format: U6 index This field contains an index value which selects the VB state being defined. Value: [0,32] — Name:
	25:23	**Reserved** Project: All Format: MBZ
	22:16	**Memory Object Control State** Project: All Format: MEMORY_OBJECT_CONTROL_STATE Specifies the memory object control state for this vertex buffer.
	15	**Reserved** Project: All Format: MBZ
	14	**Address Modify Enable** Project: All If set, the Buffer Starting Address field is used to update the state of this buffer. If clear, that field is ignored and the previously-programmed value is maintained.
	13	**Null Vertex Buffer** Project: All Format: Enable This field enabled causes any fetch for vertex data to return 0. Programming Notes — Project

VERTEX_BUFFER_STATE

		VERTEX_BUFFER_STATE.Null Vertex Buffer must be set when the VERTEX_BUFFER_STATE.Buffer Size is 0x0. — CHV, BSW
	12	**Reserved** Project: — All Format: — MBZ
	11:0	**Buffer Pitch** Format: — U12 Count of bytes This field specifies the pitch in bytes of the structures accessed within the VB. This information is required in order to access elements in the VB via a structure index. [0,2048] — Bytes — CHV, BSW **Programming Notes** • Different VERTEX_BUFFER_STATE structures can refer to the same memory region using different Buffer Pitch values. • See note on 64-bit float alignment in Buffer Starting Address.
1..2	63:0	**Buffer Starting Address** Format: — GraphicsAddress[63:0]Vertex_Buffer This field contains the byte-aligned Graphics Address LSBs of the first element of interest within the VB. Software must program this value with the combination (sum) of the base address of the memory resource and the byte offset from the base address to the starting structure within the buffer. If the Address ModifyEnable bit is clear, this field is ignored and the previous value of Buffer Starting Address for this buffer is maintained. **Programming Notes** • 64-bit floating point values must be 64-bit aligned in memory, or UNPREDICTABLE data will be fetched. When accessing an element containing 64-bit floating point values, the Buffer Starting Address and Source Element Offset values must add to a 64-bit aligned address, and BufferPitch must be a multiple of 64-bits. • VBs can only be allocated in linear (not tiled) graphics memory. • As computed index values are, by definition, interpreted as unsigned values, there is no issue with accesses to locations before (lower address value) the start of the buffer. However, these wrapped indices are subject to Max Index checking (see below).
3	31:0	**Buffer Size** Format: — U32 Count of bytes This field specifies the size of the buffer in bytes. Vertex element accesses which straddle or go past the end of the buffer will return 0's for all elements.Note that BufferSize=0 indicates that there is no valid data in the buffer. Value — Name

VERTEX_BUFFER_STATE			
		[0, FFFFFFFFh]	

VERTEX_ELEMENT_STATE

VERTEX_ELEMENT_STATE

Project:	All
Source:	RenderCS
Size (in bits):	64
Default Value:	0x00000000, 0x00000000

Description	Project
This structure is used in 3DSTATE_VERTEX_ELEMENTS to set the state associated with a vertex element. A vertex element is defined as an entity supplying from 1 to 4 DWord vertex components to be stored in the vertex URB entry. The number of supported vertex elements is:	
[CHV, BSW]: 34	CHV, BSW
The VF function will use this state, and possibly the state of the associated vertex buffer, to fetch/generate the source vertex element data, perform any required format conversions, padding with zeros, and store the resulting destination vertex element data into the vertex URB entry.	

Programming Notes	Project
• The (new) 3DSTATE_VF_SGVS command is used to specify optional insertion of VertexID and/or InstanceID into the input vertex data, logically following the processing of the VERTEX_ELEMENT_STATE structures. The VFCOMP_STORE_VID/IID encodings are no longer available in VERTEX_ELEMENT_STATE. • When SourceElementFormat is set to one of the *64*_PASSTHRU formats, 64-bit components are stored in the URB without any conversion. In this case, vertex elements must be written as 128 or 256 bits, with VFCOMP_STORE_0 being used to pad the output as required. E.g., if R64_PASSTHRU is used to copy a 64-bit Red component into the URB, Component 1 must be specified as VFCOMP_STORE_0 (with Components 2,3 set to VFCOMP_NOSTORE) in order to output a 128-bit vertex element, or Components 1-3 must be specified as VFCOMP_STORE_0 in order to output a 256-bit vertex element. Likewise, use of R64G64B64_PASSTHRU requires Component 3 to be specified as VFCOMP_STORE_0 in order to output a 256-bit vertex element. • When SourceElementFormat is set to one of the *64*_PASSTHRU formats then VFCOMP_STORE_SRC must be used for every valid component. • Any SourceElementFormat of *64*_PASSTHRU cannot be used with an element which has edge flag enabled.	CHV, BSW
The SourceElementFormat needs to be a single-component format with an element which has edge flag enabled.	

DWord	Bit	Description
0	31:26	**Vertex Buffer Index** Project: CHV, BSW

VERTEX_ELEMENT_STATE

Format:	U6

This field specifies which vertex buffer the element is sourced from.

Value	Name
[0,32]	Up to 33 VBs are supported

Programming Notes
It is possible for a vertex element to include only internally-generated data (VertexID, etc.), in which case the associated vertex buffer state is ignored.

25 **Valid**

Project:	CHV, BSW
Format:	Boolean

Value	Name	Description	Project
1h	TRUE	this vertex element is used in vertex assembly	All
0h	FALSE	this vertex element is not used.	All

24:16 **Source Element Format**

Project:	All
Format:	SURFACE_FORMAT [CHV, BSW]

Range: Valid formats are found in the 3D Primitive Processing FormatConversion portion of the vertex fetch chapter.
Format: The encoding of this field is identical the Surface Format field of the SURFACE_STATE structure, as described in the Sampler chapter.
This field specifies the format in which the memory-resident source data for this particular vertex element is stored in the memory buffer. This only applies to elements stored with VFCOMP_STORE_SRC component control. (All other component types have an explicit format).

15 **Edge Flag Enable**

Project:	CHV, BSW
Format:	Enable

Description	Project
When ENABLED, the source element is interpreted as an EdgeFlag for the vertex. If the source element is zero, the EdgeFlag will be set to FALSE. If the source element is non-zero, the EdgeFlag will be set to TRUE. The EdgeFlag bit will travel down the fixed function pipeline along with the vertex handle, etc. and not be stored in the vertex data like the other vertex elements. Refer to the fixed function descriptions for how this EdgeFlag affects rendering. Edge flags are supported for the following primitive topology types only, otherwise EdgeFlagEnable must not be ENABLED. • 3DPRIM_TRILIST*	

VERTEX_ELEMENT_STATE

- 3DPRIM_TRISTRIP*
- 3DPRIM_TRIFAN*
- 3DPRIM_POLYGON

If this bit is DISABLED for all valid VERTEX_ELEMENTs, the vertex will be assigned a default EdgeFlag of TRUE.

Edge flags are supported for all primitive topology types.	CHV, BSW

Programming Notes

- This bit must only be ENABLED on the last valid VERTEX_ELEMENT structure.
- When set, Component 0 Control must be set to VFCOMP_STORE_SRC, and Component 1-3 Control must be set to VFCOMP_NOSTORE.

14:12 **Reserved**

Project:	All
Format:	MBZ

11:0 **Source Element Offset**

Project:	All
Format:	U12 byte offset

Byte offset of the source vertex element data in the structures comprising the vertex buffer.

Value	Name
[0,2047]	

Programming Notes

See note on 64-bit float alignment in Buffer Starting Address.

1 31 **Reserved**

Project:	All
Format:	MBZ

30:28 **Component 0 Control**

Project:	All
Format:	3D_Vertex_Component_Control [CHV, BSW]

Refer to the 3D_Vertex_Component_Control table below

27 **Reserved**

Project:	All
Format:	MBZ

26:24 **Component 1 Control**

Format:	3D_Vertex_Component_Control [CHV, BSW]

VERTEX_ELEMENT_STATE

		Refer to the 3D_Vertex_Component_Control table below
	23	**Reserved** Project: All Format: MBZ
	22:20	**Component 2 Control** Format: 3D_Vertex_Component_Control [CHV, BSW] Refer to the 3D_Vertex_Component_Control table below
	19	**Reserved** Project: All Format: MBZ
	18:16	**Component 3 Control** Format: 3D_Vertex_Component_Control [CHV, BSW] Refer to the 3D_Vertex_Component_Control table below
	15:8	**Reserved** Project: All Format: MBZ
	7:0	**Reserved** Project: CHV, BSW Format: MBZ

Vertical Line Stride Override Message Descriptor Control Field

MDC_VLSO - Vertical Line Stride Override Message Descriptor Control Field

Project: CHV, BSW
Source: PRM
Size (in bits): 3
Default Value: 0x00000000

DWord	Bit	Description
0	2	**Vertical Line Stride Override** Project: All Format: Enable If set, override the Vertical Line Stride and Vertical Line Stride Offset fields in the surface state with the fields below.
	1	**Vertical Line Stride** Project: All Format: U1 Specifies number of lines (0 or 1) to skip between logically adjacent lines - provides support of interleaved (field) surfaces as textures.
	0	**Vertical Line Stride Offset** Project: All Format: U1 Specifies the offset of the initial line from the beginning of the buffer. Ignored when Override VerticalLine Stride is 0.

VFE_STATE_EX

VFE_STATE_EX

Project: CHV, BSW
Source: RenderCS
Size (in bits): 256
Default Value: 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000, 0x00000000

DWord	Bit	Description
0	31:8	**Reserved**
	7:0	**Reserved** Format: MBZ
1	31:0	**VFE Control** This field is used by VFE depending on the mode of operation. See the following tables for details. If VFE Mode = AVC-IT or AVC-MC, this field is valid as defined in Table 1 13. If VFE Mode = VC1-IT, this field is valid as defined in Table 1 14. Otherwise, this field is reserved.
2	31:0	**Interface Descriptor Remap Table** This field contains the interface descriptor remap table entries for the first 8 kernel indices. Each table entry has 4 bits, providing a remapping range of [0, 15]. The input of this table is the Interface Descriptor Offset within the MEDIA_OBJECT or MEDIA_OBJECT_EX command. As the table is limited to map the first 16 values, any Interface Descriptor Offset greater than 15 is not remapped. Bits 31:28: Remap for index = 7 Bits 27:24: Remap for index = 6 Bits 23:20: Remap for index = 5 Bits 19:16: Remap for index = 4 Bits 15:12: Remap for index = 3 Bits 11:8: Remap for index = 2 Bits 7:4: Remap for index = 1 Bits 3:0: Remap for index = 0
3	31:0	**Interface Descriptor Remap Table (cont)** This field contains the interface descriptor remap table entries for the next 8 kernel indices (index = 8...15). Each table entry has 4 bits, providing a remapping range of [0, 15]. Bits 31:28: Remap for index = 15 Bits 27:24: Remap for index = 14 Bits 23:20: Remap for index = 13 Bits 19:16: Remap for index = 12 Bits 15:12: Remap for index = 11 Bits 11:8: Remap for index = 10 Bits 7:4: Remap for index = 9 Bits 3:0: Remap for index = 8
4	31	**Scoreboard Enable** Project: CHV, BSW This field enables and disables the hardware scoreboard in the Media Pipeline. If this field is cleared, hardware ignores the following scoreboard state fields. This should be enabled at all times in the state and the scoreboard enable field in the MEDIA_OBJECT command should be use instead. If this field is disabled, the scratch space pointer calculation will be incorrect and any attempt to use the scoreboard later will result in a hardware hang. Value 0: Scoreboard disabled

VFE_STATE_EX

		1 — Scoreboard enabled
	30	**Scoreboard Type** Project: CHV, BSW This field selects the type of scoreboard in use. This field must be zero (stalling scoreboard) Value 0 — Stalling Scoreboard Value 1 — Reserved (for Non-stalling scoreboard)
	29:8	**Reserved** Format: MBZ
	7:0	**Scoreboard Mask** Project: CHV, BSW Format: Boolean Each bit indicates the corresponding dependency scoreboard is enabled. The scoreboard is based on the relative (X, Y) distance from the current threads' (X, Y) position. Value [0,7] — Name: Bit n — Description: Score n is enabled
5	31:28	**Scoreboard 3 Delta Y** Project: CHV, BSW Format: S3 Relative vertical distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.
	27:24	**Scoreboard 3 Delta X** Project: CHV, BSW Format: S3 Relative horizontal distance of the dependent instance assigned to scoreboard 3, in the form of 2's compliment.
	23:16	**Scoreboard 2 Delta (X, Y)** Project: CHV, BSW
	15:8	**Scoreboard 1 Delta (X, Y)** Project: CHV, BSW
	7:0	**Scoreboard 0 Delta (X, Y)** Project: CHV, BSW
6	31:24	**Scoreboard 7 Delta (X, Y)**

VFE_STATE_EX			
		Project:	CHV, BSW
	23:16	**Scoreboard 6 Delta (X, Y)**	
		Project:	CHV, BSW
	15:8	**Scoreboard 5 Delta (X, Y)**	
		Project:	CHV, BSW
	7:0	**Scoreboard 4 Delta (X, Y)**	
		Project:	CHV, BSW
7	31:0	**Reserved**	
		Format:	MBZ

VP8 Encoder StreamOut Format

VP8 Encoder StreamOut Format		
Project:	CHV, BSW	
Source:	VideoCS	
Size (in bits):	128	
Default Value:	0x00000000, 0x00000000, 0x00000000, 0x00000000	
DWord	**Bit**	**Description**
0	31:24	**MbY** Format: U8
	23:16	**MbX** Format: U8
	15:8	**MbClock16** Format: U8
	7:3	**Reserved** Format: MBZ
	2	**MbRcFlag** Format: U1
	1	**MBLevelInterMBConformanceFlag** Format: U1
	0	**MBLevelIntraMBConformanceFlag** Format: U1
1	31:29	**Reserved** Format: MBZ
	28:16	**MB_Residual_BitCount** Format: U13
	15:13	**Reserved** Format: MBZ
	12:0	**MB_Total_BitCount** Format: U13
2	31:25	**Reserved** Format: MBZ
	24:0	**Cbp** Format: U25
3	31	**Reserved** Format: MBZ

VP8 Encoder StreamOut Format			
	30	**LastMbFlag**	
		Format:	U1
	29	**IntraMBFlag**	
		Format:	U1
	28:24	**MBType5Bits**	
		Format:	U5
	23:19	**Reserved**	
		Format:	MBZ
	18	**QindexClampHigh**	
		Format:	U1
	17	**QindexClampLow**	
		Format:	U1
	16	**CoeffClampStatus**	
		Format:	U1
	15:0	**Reserved**	
		Format:	MBZ

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 2e: Additional Command Reference Information

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

You are free to Share — to copy, distribute, display, and perform the work

Under the following conditions:

Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).

No Derivative Works. You may not alter, transform, or build upon this work

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

GenLC Registers **1**

Memory Map **2**

Relocatable Address Ranges 3

Legacy Video Area (A_0000h – B_FFFFh) 3

VGA Paged 3

GTT Stolen Memory (GSM) 4

GTT Table Format Change 4

Graphics Stolen Memory 6

BAR1 – GTTMMADR (GTT and Regs) 7

BAR2 – GMADR (Gfx Aperture Memory) 8

IOBAR Indirect MMIO or GTT access 8

Register Range Assignments **9**

Register Clock Domains 9

Domain Registers 9

PCI Configuration Registers 9

Vendor ID and Device ID 10

PCICMD 11

PCISTS 12

Revision ID (RID) 12

Class Codes (CC) 13

HDR 13

GTTMMADR 14

GMADR – Gfx Memory Address Range (ie. Gfx Aperture) 15

IOBAR 16

Subsystem Vendor Identification (SSID) 16

Subsystem Identification (SID) 17

CAPPOINT 17

Interrupt Line Register (INTRLINE) 18

INTR 18

GGC 19

BDSM – Base of Data Stolen Memory 21

MSAC – Multi-Size Aperture Control 22

BGSM – Base of GTT Stolen Memory 23

MSI_CAPID 23
Message Control (MC) 24
MA 25
MD 25
VCID 26
VC 26
FD 27
PM Capabilities ID 27
PM Capabilities 28
PMCS 29
SWSMISCI 30
ASLE 31
ASLS 32
Memory Interface Registers (MIR) 32
IIR_RW – Alternate IIR Access 34
SCPD0 – Scratch Pad 0 Register 34
IER – Interrupt Enable Register 35
IIR – Interrupt Identity Register 37
IMR – Interrupt Mask Register 40
ISR – Interrupt Status Register 43
EIR – Error Identity Register 45
EMR – Error Mask Register 46
ESR – Error Status Register 47
Power Context Base Register (PCBR) 47
GTLC: SA MMIO Registers 48
Fence Registers 49
Gfx_FLSH_CNTL 50
GTFIFOCTL 51
GTLC Wake Control 51
GTLC Power Well Status 53
Render Force wake req 53
Render Force wake ack 54
Media Force wake req 54
Media Force wake ack 55
Render RC6 (Standby) counter 55

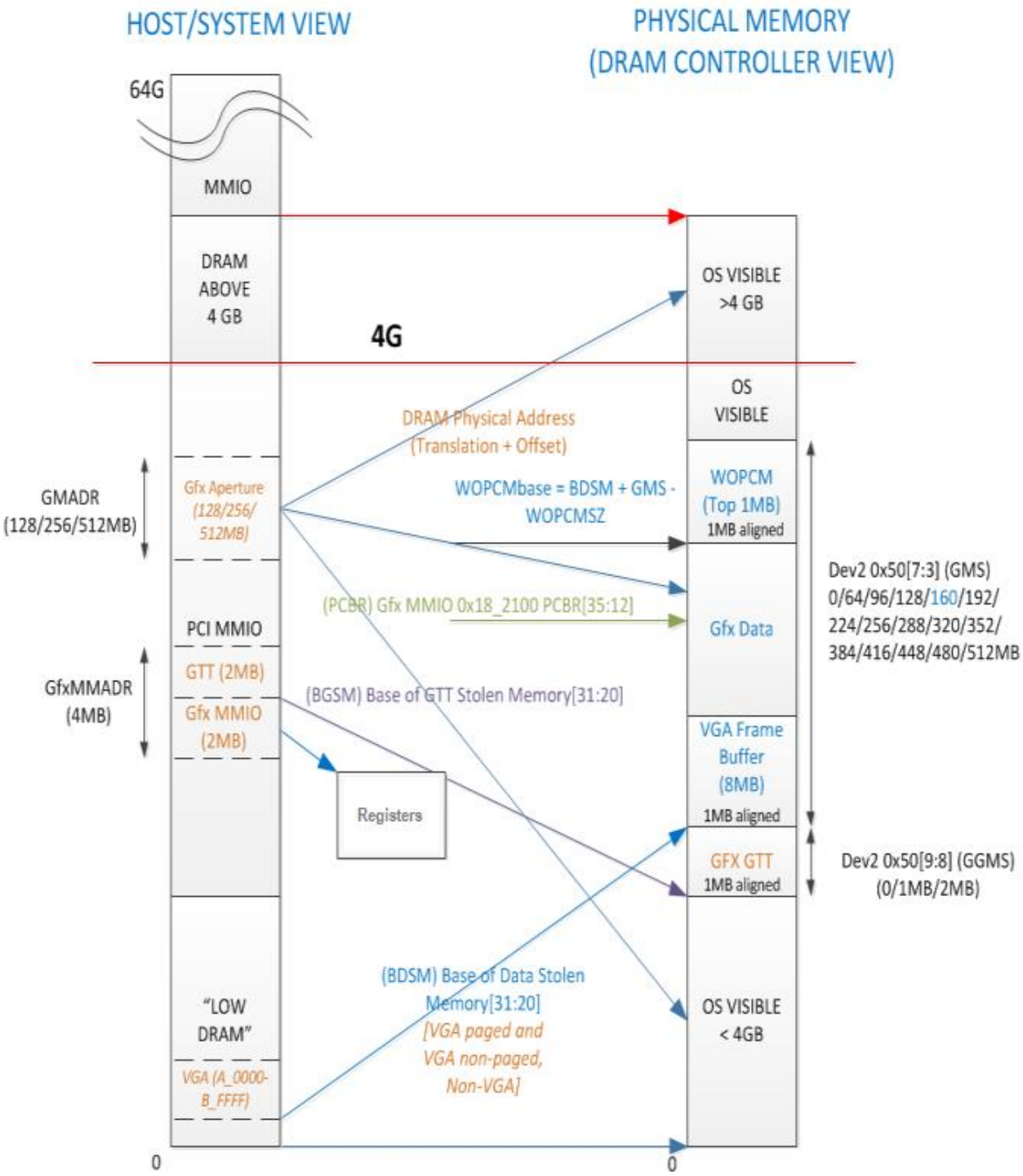
Media RC6 (Standby) counter....56
GTLC: "Render/Media/PM" Interrupt Architecture....56
GTLC: Master Interrupt Register....57
GTLC: "Render/Media" Interrupts....58
ISR – Interrupt Status Register....59
GTLC: "Power Management" Interrupts....63
GTLC: Scratch Registers....68
SWF – Software Flags....68
GTSCRATCH – GT Scratch registers....68

GenLC Registers

The figure below shows the different types of registers that reside within the GenLC block.

Memory Map

The diagram below shows the memory map for Gfx Device 2.



Relocatable Address Ranges

Bus 0, Device 2, Function 0:

1. **IOBAR** – IO access window for internal graphics. Though this window address/data register pair, using I/O semantics, the IGD and internal graphics instruction port registers can be accessed. Note, this allows accessing the same registers as GTTMMADR. The IOBAR can be used to issue writes to the GTTMMADR or the GTT table.
2. **GMADR** – Internal graphics translation window (128MB, 256MB, 512MB window).
3. **GTTMMADR** – This register requests a 4MB allocation for combined Graphics Translation Table Modification Range and Memory Mapped Range. GTTADR will be at GTTMMADR + 2MB while the MMIO base address will be the same as GTTMMADR.
4. **GGC.GMS** – Graphics Mode Select. Used to select the amount of main memory that is pre-allocated to support the internal graphics device in VGA (non-linear) and Native (linear) modes.
5. **GGC.GGMS** – GTT Graphics Memory Size. Used to select the amount of main memory that is pre-allocated to support the Internal Graphics Translation Table. (0-2MB options).

VGA memory and VGA IO addresses are also supported.

Legacy Video Area (A_0000h – B_FFFFh)

The legacy 128KB VGA memory range, frame buffer, (000A_0000h – 000B_FFFFh) can be mapped to Device #2. The VGA memory space is a 128KB space. The space's virtual address range, as seen on the IOSF bus, is A0000h-BFFFFh. The space's starting physical address is given by the Base of Data Stolen Memory register value (Dev 2 register BDSM[31:20]).

Table: Frame Buffer Accesses (VGA Memory Map Mode Register)

VRH_HT_GR06[3:2]	A0000h - AFFFFh	B0000h - B7FFFh	B8000h - BFFFFh
00	X	X	X
01	X		
10		X	
11			X

VGA Paged

The VGA frame buffer is 8MB (similar to client). The space's virtual address range, as seen on the IOSF bus, is A0000h-AFFFFh.

GTT Stolen Memory (GSM)

The base of the GTT table memory is contained in register BGSM. GSM is allocated to store the GFX translation table entries. This space is allocated to store accesses as page table entries get updated through virtual GTTMMADR range. Hardware is responsible to map PTEs into this physical space. Direct accesses to GSM are not allowed. The memory map programming and North Cluster IMR programming prevents direct GSM accesses. Only hardware translations and fetches can be directed to GSM.

ValleyView North Cluster has IMR registers that define stolen memory regions. The GSM will be mapped into an IMR region. The SAI attribute, driven on PFI, ensures the North Cluster does not invalidate GSM directed translations or fetches.

GTT Table Format Change

The GTT entry will be redefined to use bit 2 to indicate whether a page is snooped or not.

Only the Graphics-initiated accesses need this field; IOSF Host initiated cycles are already coherent, so the aperture translation path can ignore this GTT entry bit.

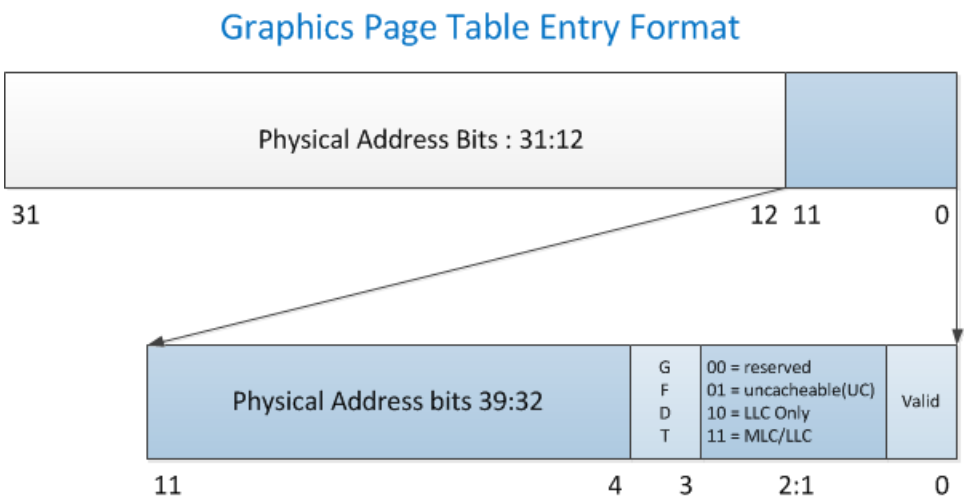


Table: GTT Table Entry Description

Bit	Description
31: 12	**Physical Page Address 31:12:** If the Valid bit is set, This field provides the page number of the physical memory page backing the corresponding Graphics Memory page.
11:4	Physical Start Address Extension: This field specified Bits 39:32 of the page table entry. This field must be zero for 32 bit addresses.
3	Graphics Data Type (GFDT) This field contains the GFDT bit for this surface when writes occur. GFDT can also be set by various state commands and indirect objects. The effective GFDT is the logical OR of the GTT entry with this field. This field is ignored for reads. Format = U1 0: No GFDT (default) 1: GFDT
2:1	Coherency Control (when GT is a non-caching agent – VLV LCIA) This field controls coherency with the CPU core caches 0x: data in this page is not coherent with the CPU caches 1x: data accesses in this page must be snooped in the CPU caches
0	**Valid PTE:** This field indicates whether the mapping of the corresponding Graphics Memory page is valid. 1: Valid 0: Invalid. An access (other than a CPU Read) through an invalid PTE will result in Page Table Error (Invalid PTE).

Graphics Stolen Memory

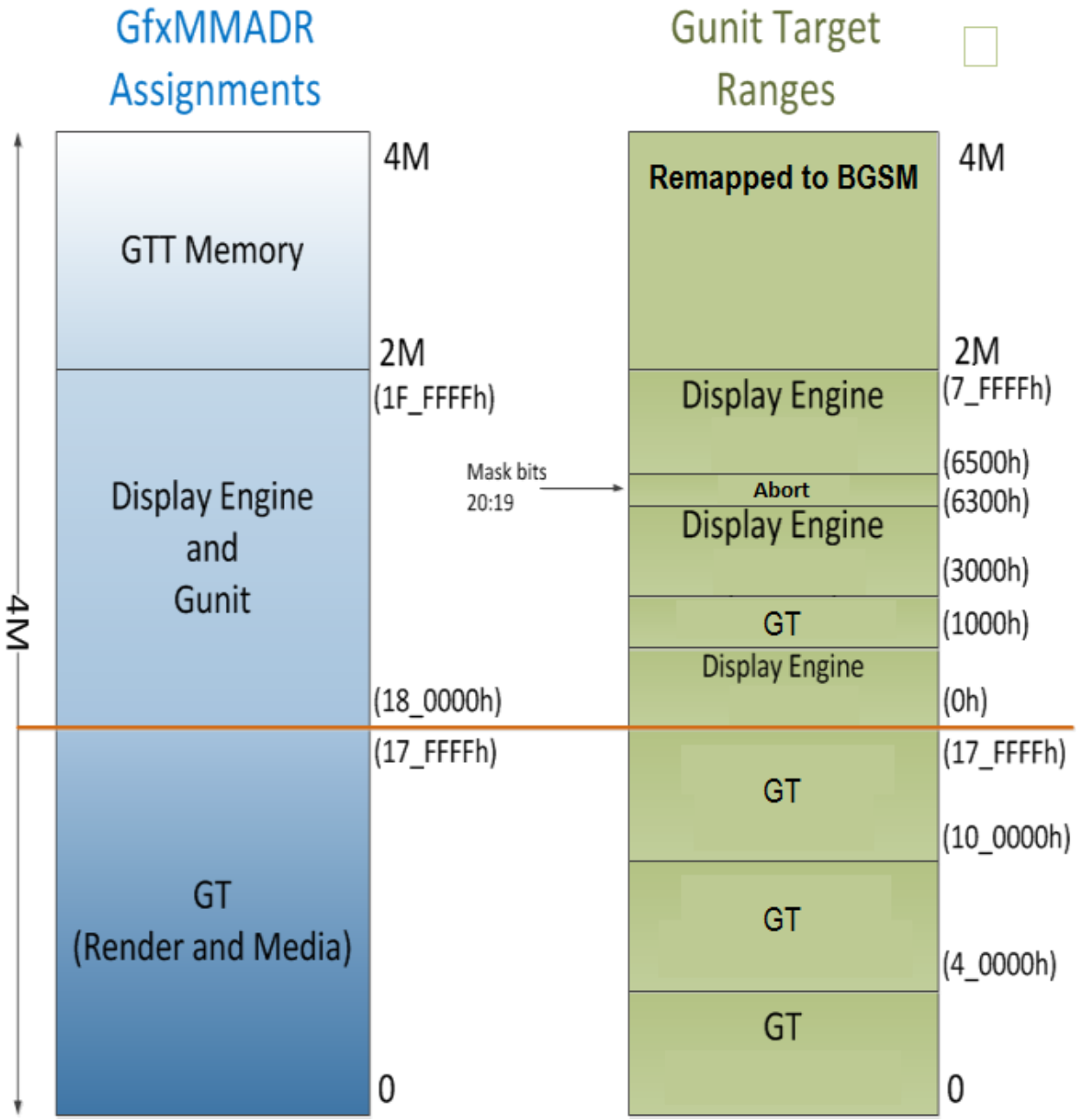
The Gen graphics engine relies on a sections of memory that are accessible to graphics but not to CPU software. This is called *stolen memory* or sometimes just *graphics memory*. This consists of 2 ranges, set up by BIOS:

1. GTT stolen memory: Programmable 1MB or 2MB space, used to store virtual->physical graphics translation tables. CPU software cannot access this directly; a section of the GTTMMADR BAR accessible to the Device 2 driver is mapped to this range.
    a. TLB fetch requests are mapped to this range
2. Data stolen memory: Programmable 32MB-512MB space, used for general graphics data (including frame buffers and other local structures). This range will be the target of some of the virtual address pages via the GTT.
    a. Lower 8MB – VGA frame buffer. VGA client can only access this region

For VLV, the Northbridge "IMR" mechanism will be used to reserve these 2 spaces for graphics hidden from the OS.

BAR1 – GTTMMADR (GTT and Regs)

This range will be marked as UC (Un-cacheable). This BAR is split in half. The bottom portion is registers while the top portion is the window to the GTT (Graphics Translation Table).



BAR2 – GMADR (Gfx Aperture Memory)

IOSF primary accesses targeting GMADR range are translated using the GTT translation table. The resulting physical addressed request is then sent to system memory. The GMADR range can be subdivided using the fence registers within Gfx MMIO space. TileY, TileX, and linear formats will be supported.

IOBAR Indirect MMIO or GTT access

The graphics interface (Device 2) contains an IOBAR register. If Device 2 is enabled, then IGD registers or the GTT table can be accessed using this IOBAR. The IOBAR is composed of an index register and a data register.

MMIO_Index: MMIO_INDEX is a 32 bit register. An IO write to this port loads the offset of the MMIO register or offset into the GTT that needs to be accessed. An IO Read returns the current value of this register. See IOBAR rules for detailed information.

MMIO_Data: MMIO_DATA is a 32 bit register. An IO write to this port is re-directed to the MMIO register pointed to by the MMIO-index register. An IO read to this port is re-directed to the MMIO register pointed to by the MMIO-index register.

The result of accesses through IOBAR can be:

a. Accesses directed to the GTT table. (i.e. route to DRAM)
b. Accesses to Graphics MMIO registers.

Note: GTT table space writes (GTTADR) are supported through this mapping mechanism.

This mechanism to access internal graphics MMIO registers must not be used to access VGA IO registers which are mapped through the MMIO space. VGA registers must be accessed directly through the dedicated VGA IO ports.

Register Range Assignments

- Gfx registers:
  - PCI Configuration registers (housed in Gfx)
  - Gfx "MIR" MMIO registers
    - Gfx related registers
    - Display "MIR" MMIO registers
      1. Housed in Gfx as a service to Display.
  - Gfx "GTLC" MMIO registers
    - SA,
    - GTLC interrupt registers
- GTLC MMIO registers
- Display MMIO registers

Register Clock Domains

This topic is still under development.

Domain Registers

PCI Configuration Registers

Table: PCI Configuration Register Summary

Name	Address	Functionality	Notes
DID	0x0000	PCI Device and Vendor ID Register	
PCICMD	0x0004	PCI Command and Status Register	
PCISTS	0x0006		
RIDCC	0x0008	Revision Identification and Class Codes	
HDR	0x000C	Header Type	
GTTMMADR	0x0010	BAR for Memory Mapped Registers and GTT Table	
GMADR	0x0018	BAR for Gfx Aperture	
IOBAR	0x0020	I/O BAR, BAR for the MMIO_INDEX and MMIO_DATA registers	Used only by SBIOS
SSID	0x002C	Subsystem Identifiers.	Only the BIOS can write this register, and only once after reset. After the first write, this register becomes read-only
CAPPOINT	0x0034	Capabilities Pointer	
INTR	0x003C	Interrupt	This register is programmed by SBIOS. It is not used by the graphics/display driver

Name	Address	Functionality	Notes
MGGC	0x0050	Graphics Control	Used to select the amount of memory pre-allocated to support the graphics device in VGA (non-linear) and Native (linear) modes
BDSM	0x005c	Base of Gfx Data Stolen Memory	Contains the VGA frame buffer and other Gfx data and command structures.
MSAC	0x0062	Multi-Size Aperture Control	
BGSM	0x0070	Base of GTT Stolen Memory	
MSI_CAPID	0x0090	Message Signaled Interrupts Capability ID and Control Register	
MA	0x0094	Message Address	Used for upstream IOSF Primary messages
MD	0x0098	Message Data	Data used for upstream IOSF Primary MSI messages
VCID	0x00B0	Vendor Capability ID	
VC	0x00B4	Vendor Capabilities	
FD	0x00C4	Functional Disable	Used by SBIOS, not by driver
PMCAP	0x00D0	Power Management Capabilities	
PMCS	0x00D4	Power Management Control/Status	Driver and SBIOS don't use this register
SWSMISCI	0x00E0	Software SMI or SCI	The graphics driver writes to this register as a means to interrupt the SBIOS
ASLE	0x00E4	System Display Event Register	SBIOS writes this register to generate an interrupt to the graphics/display driver
ASLS	0x00FC	ASL Storage	

Vendor ID and Device ID

PCI Address: 00h

Description: D2: PCI Device ID and Vendor ID Register

Field Name	Bit	Access	Reset Value	Description
DEVICE_IDENTIFICATION_NUMBER	31:16	RO	0F31	DID: Identifier assigned to the Device 2 Graphics PCI device.
VENDOR_IDENTIFICATION_NUMBER	15:0	RO	8086h	VID: PCI standard identification for Intel

PCICMD

PCI Address: 04h

Description: PCI Command Register

Field Name	Bit	Access	Reset Value	Description
Reserved	15:11	RO	00h	Reserved
INTERRUPT_DISABLE	10	RW	0b	**ID:** 0: Legacy interrupt message is enabled. 1: Disables legacy interrupt message generation on IOSF Sideband. **Note:** The interrupt status is not blocked from being reflected in PCISTS.IS. **Note:** MSI interrupt generation :(PCISTS.IS & PCICMD.BME & MSI_CAPID.MSIE) changes from 0 to 1. **Note:** Message bus interrupt assert is sent: (PCISTS.IS & ~PCICMD.ID & ~MSI_CAPID.MSIE) changes from 0 to 1. **Note:** Message bus interrupt de-assert is sent: (PCISTS.IS & ~PCICMD.ID & ~MSI_CAPID.MSIE) changes from 1 to 0
Reserved	9:3	RO	00h	Reserved
BUS_MASTER_ENABLE	2	RW	0b	**BME:**(BME & MAE are observed. But context save/restore can occur.) 0: Blocks the sending of MSI interrupts. 1: Permits the sending of MSI interrupts
Reserved	1	RW	0b	Reserved
IO_SPACE_ENABLE	0	RW	0b	**IOSE:** 0: I/O space is disabled. IORD and IOWr cycles will not be claimed. 1: I/O space is enabled. VGA_IO and Gfx_IOBAR are checked. If an IORD/IOWR matches (VGA IO address range or GFX_IOBAR), the cycle will be claimed. Care should be taken in setting up GFX_IOBAR that more than 1 match is not made as this will result in unpredictable behavior. VGA_IO: Address[15:0] is checked to determine if it falls in the VGA IO range. (The VGA IO range is 03B0h –03BBh and 03C0h – 03DFh.) Gfx_IOBAR: Address[15:3] is compared to GFX_IOBAR[15:3].

PCISTS

PCI Address: 06h

Description: PCI Status Register

Field Name	Bit	Access	Reset Value	Description
Reserved	15:5	RO	000h	Reserved
CAPABILITY_LIST	4	RO	1b	CAP: Indicates that the CAPPOINT register at 34h provides an offset into PCI Configuration Space containing a pointer to the location of the first item in the list
INTERRUPT_STATUS	3	RO	0b	IS: Reflects the state of the interrupt in the graphics device. 1: Determined by IIR and IER memory interface registers. Indicates a device 2 (Display/Media/Render) interrupt request. 0: No interrupt pending. Only when the Interrupt Disable bit in the command register is a 0 and this Interrupt Status bit is a 1, will the assert_IntX message be sent.
Reserved	2:0	RO	000b	Reserved

Revision ID (RID)

PCI Address: 08h

Description: Revision Identification

Field Name	Bit	Access	Reset Value	Description
REVISION_ID	7:0	RO	From metal straps	RID: The value in this field reflects the value of Revision ID. This register is read-able by any agent.

Class Codes (CC)

PCI Address: 09h

Description: Class Codes

Field Name	Bit	Access	Reset Value	Description
BASE_CLASS_CODE	23:16	RO	03h	**BCC:** This is an 8-bit value that indicates the base class code. When MGGC0[VAMEN] is 0 this code has the value 03h, indicating a Display Controller. When MGGC0[VAMEN] is 1 this code has the value 04h, indicating a Multimedia Device.
SUB_CLASS_CODE	15:8	RO	00h	When MGGC0[VAMEN] is 1, this value is 80h, indicating other multimedia device. When MGGC0[VAMEN] is 0 this value will be determined based on GGC register, GMS and IVD fields. When cfg_MGGC[17] = 1 or cfg_MGGC[22:20] = 3'b000 this value is 80h, otherwise its 00h 00h: VGA compatible 80h: Non VGA (GMS = "00h" or IVD = "1b")
PROGRAMMING_INTERFACE	7:0	RO	00h	**PI:** When MGGC0[VAMEN] is 0 this value is 00h, indicating a Display Controller. When MGGC0[VAMEN] is 1 this value is 00h, indicating a NOP.

HDR

PCI Address: 0Ch

Description: Header Type

Field Name	Bit	Access	Reset Value	Description
Reserved	31:24	RO	00h	Reserved
MULTI_FUNCTION_STATUS	23	RO	0b	MFUNC: Integrated graphics is a single function
HEADER_CODE	22:16	RO	00h	HDR: Indicates a type 0 configuration space header format
Reserved	15:0	RO	0000h	Reserved

GTTMMADR

PCI Address: 10-17h

Description: Gfx Memory Mapped Address Range. This is the base address for all memory mapped registers and GTT table.

This register requests allocation for the combined Graphics Translation Table Modification Range and Memory Mapped Range. The range requires 4 MB combined for MMIO and Global GTT aperture, with 512K of that used by MMIO and 2MB used by GTT. GTTADR will begin at (GTTMMADR + 2 MB) while the MMIO base address will be the same as GTTMMADR.

For the Global GTT, this range is defined as a memory BAR in graphics device config space. It is an alias into which software is required to write Page Table Entry values (PTEs). Software may read PTE values from the global Graphics Translation Table (GTT). PTEs cannot be written directly into the global GTT memory area.

The allocation is for 4MB and the base address is defined by bits [31:22].

Field Name	Bit	Access	Reset Value	Description
Reserved	63:36	RO	000000h	**Reserved for Memory Base Address (RSVD):** Must be set to 0 since addressing above 64GB is not supported.
Reserved	35:32	RO	0h	**Reserved – since 32 bit BAR.**
MBA	31:22	RW	0000h	**Memory Base Address (MBA):** 4MB combined for MMIO and Global GTT table aperture (2MB for MMIO and 2 MB for GTT).
Reserved	21:4	RO	00000h	**Reserved (RSVD):** Hardwired to 0's to indicate at least 4MB address range.
Reserved	3	RO	0b	**Prefetchable Memory (PREFMEM):** Hardwired to 0 to prevent prefetching.
MEMTYP	2:1	RO	00b	**Memory Type (MEMTYP):** 00: To indicate 32 bit base address 01: Reserved 10: To indicate 64 bit base address
Reserved	0	RO	0b	**Memory/IO Space (MIOS):** Hardwired to 0 to indicate memory space.

GMADR – Gfx Memory Address Range (ie. Gfx Aperture)

PCI Address: 18-1Fh

Description: Gfx Aperture location.

GMADR is a Prefetchable range in order to apply USWC attribute (from the processor point of view) to that range. The USWC attribute is used by the processor for write combining.

Accesses to this range will be translated to DRAM Physical memory addresses. Fence registers may be used to sub-divide this range and allow tiled surfaces (determined by fence registers).

The following sizes are supported: 128MB, 256MB, 512MB. (Determined by the MSAC register).

Field Name	Bit	Access	Reset Value	Description
Reserved	63:36	RO	00000000h	**Memory Base Address (MBA2):** Memory Base Address (MBA): Set by the OS, these bits correspond to address signals [63:36].
Reserved	35:32	RO	0h	**Reserved – 32 bit BAR supported**
MBA	31:29	RW	0000000b	**Memory Base Address (MBA):** Memory Base Address (MBA): Set by the OS, these bits correspond to address signals [31:29].
ADMSK512	28	RW-L	0b	**512MB Address Mask (512ADMSK):** This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO), depending on the value of MSAC[2:1]. See MSAC (Dev2, Func 0, offset 62h) for details.
ADMSK256	27	RW-L	0b	**256 MB Address Mask (256ADMSK):** This bit is either part of the Memory Base Address (R/W) or part of the Address Mask (RO), depending on the value of MSAC[2:1]. See MSAC (Dev 2, Func 0, offset 62h) for details.
Reserved	26:4	RO	0000000h	**Address Mask (ADM):** Hardwired to 0s to indicate at least 128MB address range.
PREFMEM	3	RO	1b	**Prefetchable Memory (PREFMEM):** Hardwired to 1 to enable prefetching.
MEMTYP	2:1	RO	00b	**Memory Type (MEMTYP):** 00: To indicate 32 bit base address 01: Reserved 10: To indicate 64 bit base address
Reserved	0	RO	0b	**Memory/IO Space (MIOS):** Hardwired to 0 to indicate memory space.

IOBAR

PCI Address: 20-23h

Description: I/O Base Address. This is used only by SBIOS. This register is the base address for the MMIO_INDEX and MMIO_DATA registers

This register provides the Base offset of the I/O registers within Device #2. Access to the 8Bs of IO space is allowed in PM state D0 when IO Enable (PCICMD bit 0) set. Access is disallowed:

- in PM states D1-D3 or
- if IO Enable is clear or
- if Device #2 is turned off or
- if Internal graphics is disabled thru the fuse or fuse override mechanisms.

Note that access to this IO BAR is independent of VGA functionality within Device #2.

If accesses to this IO bar is allowed then the GMCH claims all 8, 16 or 32 bit IO cycles from the CPU that falls within the 8B claimed.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:16	RO	0000h	Reserved
BASE_ADDRESS	15:3	RW	0000h	**BA:** Set by the OS, these bits correspond to address signals [15:3]. The IOBAR is to be used for both GTLC register programming and GTT table programming. This is an indirect access method.
Reserved	2:1	RO	0h	Reserved
RESOURCE_TYPE_RTE	0	RO	1b	Indicates a request for I/O space

Subsystem Vendor Identification (SSID)

PCI Address: 2C-2Dh

Description: This register is used to uniquely identify the subsystem where the PCI device resides.

Field Name	Bit	Access	Reset Value	Description
SUBVID	15:0	RW-O	0000h	This value is used to identify the vendor of the subsystem. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset.

Subsystem Identification (SID)

PCI Address: 2E-2Fh

Description: This register is used to uniquely identify the subsystem where the PCI device resides.

Field Name	Bit	Access	Reset Value	Description
SUBID	15:0	RW-O	0000h	This value is used to identify the vendor of the subsystem. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset.

CAPPOINT

PCI Address: 34-37h

Description: This register points to a linked list of capabilities implemented by this device.

The capability linked list is :

(Head-34, PMCAP-D0, MSI-90, VID-B0, ... End)

Field Name	Bit	Access	Reset Value	Description
Reserved	31:8	RO	000000h	Reserved
CAPABILITIES_POINTER	7:0	RW-O	D0h	The first item in the capabilities list is at address D0h (PMCS). This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset.

Interrupt Line Register (INTRLINE)

PCI Address: 3Ch

Description: Interrupt. This register is programmed by SBIOS. It is not used by the graphics/display driver. This 8-bit register is used to communicate interrupt line routing information. It is read/write and must be implemented by the device. POST software will write the routing information into this register as it initializes and configures the system.

The value in this register tells which input of the system interrupt controller(s) the device's interrupt pin is connected to. The device itself does not use this value, rather it is used by device drivers and operating systems to determine priority and vector information.

Field Name	Bit	Access	Reset Value	Description
INTRLINE	7:0	RW	00h	ILIN: BIOS written value to communicate interrupt line routing information to the device driver Used to communicate interrupt line routing information. POST software writes the routing information into this register as it initializes and configures the system. The value in this register indicates to which input of the system interrupt controller the device's interrupt pin is connected.

INTR

PCI Address: 3Dh

Description: Interrupt. This register is programmed by SBIOS. It is not used by the graphics/display driver

Field Name	Bit	Access	*Reset Value*	Description
INTERRUPT_PIN	7:0	RO	01h	IPIN: Value indicates which interrupt pin this device uses. This field is hard coded to 1h since Valleyview Device 2 is a single function device. The PCI spec requires that it use INTA#. 01h: INTA#.

GGC

PCI Address: 50-51h

Description: GMCH Graphics Control Register

Field Name	Bit	Access	Reset Value	Description
Reserved	15	RO	0b	Reserved
VAMEN	14	RW-L	0b	Enables the use of the iGFX engines for Versatile Acceleration. 1 – iGFX engines are in Versatile Acceleration Mode. Device 2 Class Code is 048000h. 0 – iGFX engines are in iGFX Mode. Device 2 Class Code is 030000h.
Reserved	13:10	RO	0h	Reserved
GGMS	9:8	RW-L	00b	GTT Graphics Memory Size (GGMS): This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics Translation Table. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled. GSM is assumed to be a contiguous physical DRAM space with DSM, and BIOS needs to allocate a contiguous memory chunk. Hardware will drive the base of GSM from DSM only using the GSM size programmed in the register. 0h: No memory pre-allocated. GTT cycles (Mem and IO) are not claimed. 1h: 1 MB of memory pre-allocated for GTT. 2h: 2 MB of memory pre-allocated for GTT. 3h: Reserved. All unspecified encodings of this register field are reserved, hardware functionality is not guaranteed if used.
GMS	7:3	RW-L	00101b	Graphics Mode Select (GMS). This field is used to select the amount of Main Memory that is pre-allocated to support the Internal Graphics device in VGA (non-linear) and Native (linear) modes. The BIOS ensures that memory is pre-allocated only when Internal graphics is enabled.

Field Name	Bit	Access	Reset Value	Description
				Hardware does not clear or set any of these bits automatically based on IGD being disabled/enabled. BIOS Requirement: BIOS must not set this field to 0h if IVD (bit 1 of this register) is 0. 0h = 0MB 10h = 512MB 1h = Reserved (32 MB – encoding is only valid for steppings C0 and beyond.) 2h = 64MB 3h = 96MB 4h = 128MB 5h = 160MB 6h = 192MB 7h = 224MB 8h = 256MB 9h = 288MB Ah = 320MB Bh = 352MB Ch = 384MB Dh = 416MB Eh = 448MB Fh = 480MB Other = Reserved When GMS != 000 (and VD=0): When GMS == 000 : No address compare will occur against VGA memory range or the VGA IO register range. Also, CC[15:8] is changed to 8'h80 from 8'h00
Reserved	2	RO	0b	Reserved ():

Field Name	Bit	Access	Reset Value	Description
VGA_DISABLE	1	RW-L	0b	**VGA Disable (VD):** 0: Enable. Device 2 (IGD) claims VGA memory and IO cycles, the Sub-Class Code within Device 2 Class Code register is 00. 1: Disable. Device 2 (IGD) does not claim VGA cycles (Mem and IO), and the Sub- Class Code field within Device 2 function 0 Class Code register is 80. BIOS Requirement: BIOS must not set this bit to 0 if the GMS field pre-allocates no memory. This bit MUST be set to 1 if Device 2 is disabled either via a fuse or fuse override (CAPID0[38] = 1) or via a register (DEVEN[3] = 0).
GGCLCK	0	RW-L	0b	When set to 1b, this bit will lock all bits in this register.

BDSM – Base of Data Stolen Memory

PCI Address: 5C-5Fh

Description: This register contains the base address of Graphics Data Stolen DRAM memory.

Graphics Stolen Memory is within DRAM space. The base of stolen memory will always be below 4G.

Field Name	Bit	Access	Reset Value	Description
BDSM	31:20	RW-L	000h	**BDSM:** BASE_OF_Data_STOLEN_MEMORY This register contains bits 31 to 20 of the base address of Data stolen DRAM memory. For certain GTLC generated accesses, this base register will be added to the GTLC provided offset address, forming the full physical address for the PFI fabric. This is also used as a base for VGA paged accesses. The display engine also uses the register.
Reserved	19:1	RO	00000h	Reserved
Lock	0	RW-L	0b	This bit will lock all writeable settings in this register, including itself.

MSAC – Multi-Size Aperture Control

PCI Address: 62h

Description: This register determines the size of the graphics memory aperture. Only the system BIOS will write this register based on pre- boot address allocation efforts. Graphics may read this register to determine the correct aperture size. System BIOS needs to save this value on boot so that it can reset it correctly during S3 resume.

The size of the aperture must not be modified by software after its location is written into GMADR (offset 18h).

Field Name	Bit	Access	Reset Value	Description
Reserved	7:3	RO	00000b	Reserved
LHSAS	2:1	RW	01b	**Untrusted Aperture Size (LHSAS):** 00: bits [28:27] of GMADR register are made R/W allowing **128MB** of GMADR 01: bit [28] of GMADR is made R/W and bit [27] of GMADR is forced to zero allowing **256MB** of GMADR 10: Illegal programming. 11: bits [28:27] of GMADR register are made Read only and forced to zero, allowing only **512MB** of GMADR
Reserved	0	RO	0b	Reserved

BGSM – Base of GTT Stolen Memory

PCI Address: 70-73h

Description: Base of GTT table in Gfx Stolen Memory

The GTT table is located within Graphics Stolen Memory in DRAM space. The base of stolen memory will always be below 4G.

Field Name	Bit	Access	Reset Value	Description
BGSM	31:20	RW-L	000h	**BGSM:** Gfx Base of GTT Stolen Memory This register contains bits 31 to 20 of the base address of GTT Table in stolen DRAM memory. BIOS determines the base of GTT stolen memory by subtracting the GTT graphics stolen memory size (PCI Device 2 offset 50 bits 9:8) from the Graphics Base of Data Stolen Memory (PCI Device 2 offset 5C bits 31:20).
Reserved	19:1	RO	00000h	Reserved
Lock	0	RW-L	0b	This bit will lock all writeable settings in this register, including itself.

MSI_CAPID

PCI Address: 90-91h

Description: Message Signaled Interrupts Capability ID

Field Name	Bit	Access	Reset Value	Description
POINTER_TO_NEXT_CAPABILITY	15:8	RW-O	B0h	Points to the next item in the list (B0=VCID support). Next pointer changed with HSD#259253. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset.
CAPABILITY_ID	7:0	RO	05h	CAPID: Indicates an MSI capability

Message Control (MC)

PCI Address: 92-93h

Description: Message Signaled Control Register

Field Name	Bit	Access	Reset Value	Description
Reserved	15:8	RO	00h	Reserved
64_BIT_ADDRESS_CAPABLE	7	RO	0b	**C64:** 32-bit capable only
MULTIPLE_MESSAGE_ENABLE	6:4	RW	000b	**MME:** This field is RW for software compatibility, but only a single message is ever generated System software programs this field to indicate the actual number of messages allocated to this device. This number will be equal to or less than the number actually requested. The encoding is the same as for the MMC field below.
MULTIPLE_MESSAGE_CAPABLE	3:1	RO	000b	**MMC:** This device is only single message capable System Software reads this field to determine the number of messages being requested by this device. Value: Number of requests 000: 1 001- 111: Reserved
MSI_ENABLE	0	RW	0b	**MSIE:** If set, MSI is enabled and traditional interrupts are not used to generate interrupts. PCICMDSTS.BME must be set for an MSI to be generated. 0: MSI interrupts are disabled. 1: MSI interrupts are enabled. Permits sending an MSI interrupt.

MA

PCI Address: 94-97h

Description: Message Address

Field Name	Bit	Access	Reset Value	Description
ADDRESS	31:2	RW	00000000h	**MA:** Lower 32-bits of the system specified message address, always DW aligned. When the GVD issues an MSI interrupt as a MEMWR on the SCL, the memory address corresponds to the value of this field
Reserved	1:0	RO	00b	Reserved

MD

PCI Address: 98-99h

Description: Message Data

Field Name	Bit	Access	Reset Value	Description
Reserved	31:16	RO	0000h	Reserved
DATA	15:0	RW	0000h	**MD:** This 16-bit field is programmed by system software. This is forms the lower word of data for the MSI write transaction.

VCID

PCI Address: B0h

Description: Vendor Capability ID

Field Name	Bit	Access	Reset Value	Description
VERSION	31:24	RO	01h	VS: Identifies this as the first revision of the CAPID register definition
LENGTH	23:16	RO	07h	LEN: This field has the value 07h to indicate the structure length (8 bytes)
NEXT_CAPABILITY_POINTER	15:8	RW-O	00h	00 indicates capability list ends here. This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset. Write-once allowing the capability list to be changed.
CAPABILITY_ID_CID	7:0	RO	09h	Identifies this as a vendor dependent capability pointers

VC

PCI Address: B4h

Description: Vendor Capabilities

Any sku related fuses would be added here.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:1	RO	0000000h	Reserved
Reserved	0	RO	0b	Placeholder for sku related fusing. Valleyview has no need for this.

FD

PCI Address: C4h

Description: Functional Disable. This register is used by SBIOS, not by driver

Field Name	Bit	Access	Reset Value	Description
Reserved	31:1	RO	00000000h	Reserved
FUNCTION_DISABLE	0	RW	0b	**FD:** 0: Default – normal operation. 1: When set, the function is disabled (configuration space is disabled).All new requests on the IOSF Primary bus, including any new configuration cycle requests are not claimed on IOSF Primary. This bit as no effect register accessibility via IOSF SB. Once programmed to '1', the only way to re-enable device 2 is via an IOSF SB write of '0' to this register.

PM Capabilities ID

PCI Address: D0-D1h

Description: Power Management Capabilities ID

Field Name	Bit	Access	Reset Value	Description
NEXT_POINTER	15:8	RW-O	90h	Indicates the next item in the capabilities list (90=MSI). This register should be programmed by BIOS during boot-up. Once written, this register becomes Read_Only. This register can only be cleared by a Reset. Write once allowing changing of the capabilities list.
CAPABILITIES_ID	7:0	RO	01h	CAPID: SIG defines this ID is 01h for power management

PM Capabilities

PCI Address: D2-D3h

Description: Power Management Capabilities

Field Name	Bit	Access	Reset Value	Description
PME_SUPPORT	15:11	RO	00h	PMES The graphics controller does not generate PME#
D2_SUPPORT	10	RO	0b	D2S: The D2 power management state is not supported
D1_SUPPORT	9	RO	0b	D1S: The D1 power management state is not supported
Reserved	8:6	RO	000b	Reserved
DEVICE_SPECIFIC_INITIALIZATION	5	RO	1b	Hardwired to 1 to indicate that special initialization of the graphics controller is required before generic class device driver is to use it
Reserved	4:3	RO	00b	Reserved
VERSION	2:0	RO	010b	VS: Indicates compliance with revision 1.1 of the PCI Power Management Specification

PMCS

PCI Address: D4-D5h

Description: Power Management Control/Status. Driver doesn't use this register. SBIOS doesn't use this register

Field Name	Bit	Access	Reset Value	Description
Reserved	31:2	RO	00000000h	Reserved
POWER_STATE_PS	1:0	RW	00b	This field indicates the current power state of the IGD and can be used to set the IGD into a new power state. If software attempts to write an unsupported state to this field, write operation must complete normally on the bus, but the data is discarded and no state change occurs. On a transition from D3 to D0 the graphics controller is optionally reset to initial values. Behavior of the graphics controller in supported states is detailed in the power management section of the Bspec. **Bits[1:0] Power state** 00: D0 Default 01: D1 Not Supported 10: D2 Not Supported 11: D3

SWSMISCI

PCI Address: E0-E1h

Description: Software SMI or SCI

The SCI mechanism for driver / BIOS communication. SMI is a system wide lock interrupt (halts the all the cores) as opposed to SCI. Vista and Win7 recommend to use the SCI. The SMI is slowly being phased out.

This register serves 2 purposes:

1. Support selection of SMI or SCI event source (SMISCISEL – bit15)
2. Event trigger (bit 0).

To generate a SW SCI event, software (System BIOS/Graphics driver) should program bit 15 (SMISCISEL) to 1. This is typically programmed once (assuming SMIs are never triggered). On a "0" to "1" subsequent transition in bit 0 of this register (caused by a software write operation), GMCH sends a single SCI message. The SCI will set the DMISCI bit in its TCO1_STS register and TCOSCI_STS bit in its GPE0 register upon receiving this message from DMI.

Once written as 1, software must write a "0" to this bit to clear it, and all other write transitions (1->0, 0->0, 1->1) or if bit 15 is "0" will not cause GMCH to send SCI message to DMI link.

To generate a SW SMI event, software should program bit 15 to 0 and trigger an SMI.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:16	RO	0000h	Reserved
SMI_OR_SCI_EVENT_SELECT	15	RW	0b	MCS: SMI or SCI event select. 0 = SMI 1 = SCI
SOFTWARE_SCRATCH_BITS	14:1	RW	0000h	Used by driver to communicate information to SBIOS. No hardware functionality
SMI_OR_SCI_EVENT	0	RW	0b	**MCE:** MCS=1, setting this bit causes an SCI. MCS=0, setting this bit causes an SMI. A 1 to 0, 0 to 0 or 1 to 1 transition of this bit does not trigger any events. The graphics driver writes to this register as a means to interrupt the SBIOS

ASLE

PCI Address: E4-E7h

Description: System Display Event Register. SBIOS writes this register to generate an interrupt to the graphics/display driver

Field Name	Bit	Access	Reset Value	Description
ASLE_SCRATCH_TRIGGER_3	31:24	RW	00h	AST3: The writing of this by field (byte) – even if just writing back the original contents – will trigger a display controller interrupt (when the memory interface register bits IER[0] = 1 and IMR[0] = 0). If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common
ASLE_SCRATCH_TRIGGER_2	23:16	RW	00h	AST2: The writing of this by field (byte) – even if just writing back the original contents – will trigger a display controller interrupt (when the memory interface register bits IER[0] = 1 and IMR[0] = 0). If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common
ASLE_SCRATCH_TRIGGER_1	15:8	RW	00h	AST1: The writing of this by field (byte) – even if just writing back the original contents – will trigger a display controller interrupt (when the memory interface register bits IER[0] = 1 and IMR[0] = 0). If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common
ASLE_SCRATCH_TRIGGER_0	7:0	RW	00h	AST0: The writing of this by field (byte) – even if just writing back the original contents – will trigger a display controller interrupt (when the memory interface register bits IER[0] = 1 and IMR[0] = 0). If written as part of a 16-bit or 32-bit write, only one interrupt is generated in common

ASLS

PCI Address: FCh

Description: ASL Storage. The Valleyview display driver does not need this register since memory Operational Region (OpRegion) is available. This register is kept for use as scratch space.

This software scratch register only needs to be read/write accessible. The exact bit register usage must be worked out in common between System BIOS and driver software, but storage for switching/indicating up to 6 devices is possible with this amount.

For each device, the ASL control method with require two bits for _DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for _DGS (enable/disable requested), and two bits for _DCS (enabled now/disabled now, connected or not).

Field Name	Bit	Access	Reset Value	Description
SCRATCH	31:0	RW	00000000h	This register provides a means for the BIOS to communicate with the driver. This definition of this scratch register is worked out in common between System BIOS and driver software. Storage for up to 6 devices is possible. For each device, the ASL control method requires two bits for _DOD (BIOS detectable yes or no, VGA/NonVGA), one bit for _DGS (enable/disable requested), and two bits for DCS (enabled now/disabled now, connected or not).

Memory Interface Registers (MIR)

The graphics interface will claim GTTMMADR display offset range: 0x18_1000h to 0x18_2FFFh.

Table: Memory Interface Registers

Name	Address	Functionality	Notes
IIR_RW	0x18_2084	Alternate means to access the IIR register	Provides software RW access to the IIR register. Write/Restore to the IIR register using this address will not trigger an interrupt to the CPU.
SCPD0	0x18_209C	Scratch Pad 0 Register	
IER	0x18_20A0	Interrupt Enable Register	Contains an interrupt enable bit for each interrupt bit in the Interrupt Identity Register (IIR) register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources. The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set.

Name	Address	Functionality	Notes
IIR	0x18_20A4	Interrupt Identity Register	Contains the persistent values of the interrupt bits that are unmasked by the IMR and thus can generate a CPU interrupt (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. Bits set in this register will remain set until the interrupt condition is cleared by software. Writing a 1 into the appropriate bit position within this register clears interrupts.
IMR	0x18_20A8	Interrupt Mask	Used by software to control which ISR bits are masked or unmasked. Unmasked bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. Masked bits will not be reported in the IIR and therefore cannot generate CPU interrupts.
ISR	0x18_20AC	Interrupt Status	Contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts. The User Interrupt bit in this register is a short pulse therefore software should not expect to use this register to sample these conditions.
EIR	0x18_20B0	Error Identity	This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register (ISR). In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR.
EMR	0x18_20B4	Error Mask	This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.
ESR	0x18_20B8	Error Status	This register contains the non-persistent values of all hardware-detected error condition bits.
PCBR	0x18_2120	Power Context Base Register	

IIR_RW – Alternate IIR Access

MMIO Address Offset: 18_2084oh

Description: Alternate means to access the IIR register. This address provides software RW access to the IIR register. Write/Restore to the IIR register using this address will not trigger an interrupt to the CPU.

Field Name	Bit	Access	Reset Value	Description
VED_Power_Interrupt	31	RW	0b	See IIR register for definition of this bit.
Reserved	30:24	RW	0000000b	Reserved
VED_Block_Interrupt	23	RW	0b	See IIR register for definition of this bit.
ISP_Interrupt	22	RW	0b	See IIR register for definition of this bit.
LPE_PIPEB_Interrupt	21	RW	0b	See IIR register for definition of this bit.
LPE_PIPEA_Interrupt	20	RW	0b	See IIR register for definition of this bit.
MIPIB_INTERRUPT	19	RW	0b	See IIR register for definition of this bit.
MIPIA_INTERRUPT	18	RW	0b	See IIR register for definition of this bit.
DISPLAY_HOTSYNC_INTERRUPT	17	RW	0b	See IIR register for definition of this bit.
Reserved	16	RW	0b	Reserved
MASTER_ERROR	15	RW	0b	See IIR register for definition of this bit.
Reserved	14:8	RW	0000000b	Reserved
DISPLAY_PIPE_A_VBLANK	7	RW	0b	See IIR register for definition of this bit.
DISPLAY_PIPE_A_EVENT	6	RW	0b	See IIR register for definition of this bit.
DISPLAY_PIPE_B_VBLANK	5	RW	0b	See IIR register for definition of this bit.
DISPLAY_PIPE_B_EVENT	4	RW	0b	See IIR register for definition of this bit.
DISPLAY_A_DPBM	3	RW	0b	See IIR register for definition of this bit.
DISPLAY_B_DPBM	2	RW	0b	See IIR register for definition of this bit.
GAM_Write_Invalid	1	RW	0b	See IIR register for definition of this bit.
ASLE	0	RW	0b	See IIR register for definition of this bit.

SCPD0 – Scratch Pad 0 Register

MMIO Address Offset: 18_209Coh

Description: Scratch Pad 0 Register

Field Name	Bit	Access	Reset Value	Description
SCPD0	31:0	RW	00000000h	Software scratch pad

IER – Interrupt Enable Register

MMIO Address Offset: 18_20A0oh

Description: IER: Interrupt Enable Register. The IER register contains an interrupt enable bit for each interrupt bit in the Interrupt Identity Register (IIR) register. A disabled interrupt will still appear in the Interrupt Identity Register to allow polling of interrupt sources. The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the Interrupt Identity Register becomes set.

Note: The MSI enable and PCICMD "Interrupt disable bit" are additional conditioning that occurs before an actual MSI or INTA message is generated.

Field Name	Bit	Access	Reset Value	Description
VED_Power_Interrupt	31	RW	0b	1 = Generate a CPU interrupt when associated IIR bit and this bit are both set indicating a VED power interrupt. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
Reserved	30:24	RW	0000000b	Reserved
VED_Block_Interrupt	23	RW	0b	1 = Generate a CPU interrupt when associated IIR bit and this bit are both set indicating a VED hardware block interrupt. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
ISP_Interrupt	22	RW	0b	1 = Generate a CPU interrupt when associated IIR bit and this bit are both set indicating an ISP hardware block interrupt. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
LPE_PIPEB_Interrupt	21	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set indicating a MIPIB event. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
LPE_PIPEA_Interrupt	20	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set indicating a MIPIB event. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
MIPIB_INTERRUPT	19	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set indicating a MIPIB event. 0 = Do not generate a CPU interrupt

Field Name	Bit	Access	Reset Value	Description
				when associated IIR bit is set.
MIPIA_INTERRUPT	18	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set indicating a MIPIA event. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
DISPLAY_HOTSYNC_INTERRUPT	17	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set indicating a HOT Plug event. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
Reserved	16	RW	0b	Reserved
MASTER_ERROR	15	RW	0b	1 = Generate a CPU interrupt when a master error occurs. A master error condition occurs if any of the bits in the Error Identity Register (EIR) are set. 0 = Do not generate a CPU interrupt when a master error occurs.
Reserved	14:8	RW	0000000b	Reserved
DISPLAY_PIPE_A_VBLANK	7	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
DISPLAY_PIPE_A_EVENT	6	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
DISPLAY_PIPE_B_VBLANK	5	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
DISPLAY_PIPE_B_EVENT	4	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
DISPLAY_A_DPBM	3	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.

Field Name	Bit	Access	Reset Value	Description
DISPLAY_B_DPBM	2	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
GAM_Write_Invalid	1	RW	0b	1 = Generate a CPU interrupt when the associated IIR bit and this bit are both set. 0 = Do not generate a CPU interrupt when associated IIR bit is set.
ASLE	0	RW	0b	1 = Generate a CPU interrupt when the ASLE register in the PCI Device 2 configuration space is written. 0 = Do not generate a CPU interrupt when the ASLE register is written.

IIR – Interrupt Identity Register

MMIO Address Offset: 18_20A4oh

Description: The IIR register contains the persistent values of the interrupt bits that are unmasked by the IMR and thus can generate a CPU interrupt (if enabled via the IER). When a CPU interrupt is generated, this should be the first register to be interrogated to determine the source of the interrupt. Bits set in this register will remain set until the interrupt condition is cleared by software. Writing a 1 into the appropriate bit position within this register clears interrupts.

Programming Note: Prior to clearing a Display Pipe-sourced interrupt (e.g., Display Pipe A VBLANK) in the IIR, the corresponding interrupt (source) status in the PIPEASTAT register (e.g., Pipe A VBLANK Interrupt Status bit of PIPEASTAT) must first be cleared.

Note: Bits in this register can be written to '1' by software via a write to 0x18_2084.

A direct write of '1' to any set bit will clear the bit.

Field Name	Bit	Access	Reset Value	Description
VED_Power_Interrupt	31	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
Reserved	30:24	RW1C	0000000b	Reserved
VED_Block_Interrupt	23	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
ISP_Interrupt	22	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked.

Field Name	Bit	Access	Reset Value	Description
				0 = This interrupt condition has either not occurred, or was masked.
LPE_PIPEB_Interrupt	21	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. This occurs when a MIPIB interrupt event has been detected. The specific trigger of this interrupt can be read in the associated status register. 0 = This interrupt condition has either not occurred, or was masked.
LPE_PIPEA_Interrupt	20	RW1C	-b	1 = If associated ISR bit asserts and is not IMR masked. This occurs when a MIPIB interrupt event has been detected. The specific trigger of this interrupt can be read in the associated status register. 0 = This interrupt condition has either not occurred, or was masked.
MIPIB_INTERRUPT	19	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. This occurs when a MIPIB interrupt event has been detected. The specific trigger of this interrupt can be read in the associated status register. 0 = This interrupt condition has either not occurred, or was masked.
MIPIA_INTERRUPT	18	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. This occurs when a MIPIA interrupt event has been detected. The specific trigger of this interrupt can be read in the associated status register. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_HOTSYNC_INTERRUPT	17	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. This occurs when a port hotplug/unplug event has been detected. The specific trigger of this interrupt can be read in the port hotplug status register. 0 = This interrupt condition has either not occurred, or was masked.
Reserved	16	RW1C	0b	Reserved
MASTER_ERROR	15	RW1C	0b	1 = A master error has occurred. A master error condition occurs if any of

Field Name	Bit	Access	Reset Value	Description
				the bits in the Error Identity Register (EIR) are set. 0 = This interrupt condition has either not occurred, or was masked. "OR" of EIR bits.
Reserved	14:8	RW1C	0000000b	Reserved
DISPLAY_PIPE_A_VBLANK	7	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_PIPE_A_EVENT	6	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_PIPE_B_VBLANK	5	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_PIPE_B_EVENT	4	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_A_DPBM	3	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
DISPLAY_B_DPBM	2	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
GAM_Write_Invalid	1	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.
ASLE	0	RW1C	0b	1 = If associated ISR bit asserts and is not IMR masked. 0 = This interrupt condition has either not occurred, or was masked.

IMR – Interrupt Mask Register

MMIO Address Offset: 18_20A8oh

Description: The IMR register is used by software to control which ISR bits are masked or unmasked. Unmasked bits will be reported in the IIR, possibly triggering a CPU interrupt, and will persist in the IIR until cleared by software. Masked bits will not be reported in the IIR and therefore cannot generate CPU interrupts

Field Name	Bit	Access	Reset Value	Description
VED_Power_Interrupt	31	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
Reserved	30:24	RW	1111111b	Reserved
VED_Block_Interrupt	23	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
ISP_Interrupt	22	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
LPE_PIPEB_Interrupt	21	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
LPE_PIPEA_Interrupt	20	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.

Field Name	Bit	Access	Reset Value	Description
MIPIB_INTERRUPT	19	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
MIPIA_INTERRUPT	18	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
DISPLAY_HOTSYNC_INTERRUPT	17	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
Reserved	16	RW	1h	Reserved
MASTER_ERROR	15	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
Reserved	14:8	RW	1111111b	Reserved
DISPLAY_PIPE_A_VBLANK	7	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
DISPLAY_PIPE_A_EVENT	6	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.

Field Name	Bit	Access	Reset Value	Description
DISPLAY_PIPE_B_VBLANK	5	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
DISPLAY_PIPE_B_EVENT	4	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
DISPLAY_A_DPBM	3	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
DISPLAY_B_DPBM	2	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
GAM_Write_Invalid	1	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.
ASLE	0	RW	1b	1 = Masked. This interrupt will neither be reported in the IIR, nor generate a CPU interrupt. 0 = Unmasked. This interrupt will be reported in the IIR. It will generate a CPU interrupt if the corresponding bit in the IER register is set.

ISR – Interrupt Status Register

MMIO Address Offset: 18_20Acoh

Description: The ISR (Interrupt Status register) contains the non-persistent value of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR (i.e., set bits must be cleared by software). Bits in the IER are used to selectively enable IIR bits to cause CPU interrupts. The User Interrupt bit in this register is a short pulse therefore software should not expect to use this register to sample these conditions

Field Name	Bit	Access	Reset Value	Description
VED_Power_Interrupt	31	RW1S	0b	A write of '1' indications this interrupt condition currently exists. Once written '1', hardware will clear this bit the next clock. In other words, a write to this bit mimics a wire pulse.
Reserved	30:24	RO	0000000b	Reserved
VED_Block_Interrupt	23	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
ISP_Interrupt	22	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
LPE_PIPEB_Interrupt	21	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
LPE_PIPEA_Interrupt	20	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
MIPIB_INTERRUPT	19	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
MIPIA_INTERRUPT	18	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_HOTSYNC_INTERRUPT	17	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.

Field Name	Bit	Access	Reset Value	Description
Reserved	16	RO	0h	Reserved
MASTER_ERROR	15	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
Reserved	14:8	RO	0h	Reserved
DISPLAY_PIPE_A_VBLANK	7	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_PIPE_A_EVENT	6	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_PIPE_B_VBLANK	5	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_PIPE_B_EVENT	4	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_A_DPBM	3	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
DISPLAY_B_DPBM	2	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
GAM_Write_Invalid	1	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.
ASLE	0	RO	0b	1 = This interrupt condition currently exists. 0 = This interrupt condition currently does not exist.

EIR – Error Identity Register

MMIO Address Offset: 18_20B0oh

Description: Error Identity. This register contains the persistent values of ESR error status bits that are unmasked via the EMR register. The logical OR of all (defined) bits in this register is reported in the Master Error bit of the Interrupt Status Register (ISR). In order to clear an error condition, software must first clear the error by writing a 1 to the appropriate bit(s) in this field. If required, software should then proceed to clear the Master Error bit of the IIR

Field Name	Bit	Access	Reset Value	Description
Reserved	31:7	RO	0000000h	Reserved
TLB_DataE	6	RW1C	0b	Set if TLB has a data valid error.
Gunit_TLB_PTE	5	RW1C	0b	Set if TLB has a PTE translation error.
PAGE_TABLE_ERROR	4	RW1C	0b	PTE: This bit is set when a Graphics Memory Mapping Error is detected and it's not EMR masked. The cause of the error is indicated (to some extent) in the PGTBL_ER register. This error condition cannot be cleared except by reset (i.e., it is a fatal error).
Reserved	3:1	RO	0h	Reserved
CLAIM_ERROR	0	RW1C	0b	If EMR[0]=1, this bit is set when an address within the address space is accessed, but no memory mapped register exists in this address. This error condition can be cleared by writing 1b to this bit.

EMR – Error Mask Register

MMIO Address Offset: 18_20B4oh

Description: Error Mask. This register contains a bit mask that selects which error condition bits (from the ESR) are reported in the EIR.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:7	RO	0000000h	Reserved
TLB_DataE	6	RW	1b	1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.
TLB_PTE	5	RW	1b	1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.
PAGE_TABLE_ERROR	4	RW	1b	1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.
Reserved	3:1	RO	0h	Reserved
CLAIM_ERROR	0	RW	1b	1 = Masked. This error will neither be reported in the EIR, nor set the master error condition in the IIR. 0 = Unmasked. This interrupt will be reported in the EIR and will set the master error condition in the IIR.

ESR – Error Status Register

MMIO Address Offset: 18_20B8oh

Description: Error Status. This register contains the non-persistent values of all hardware-detected error condition bits.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:7	RO	0000000h	Reserved
TLB_DataE	6	RO	0b	1 = This error condition currently exists. 0 = This error condition currently does not exist.
TLB_PTE	5	RO	0b	1 = This error condition currently exists. 0 = This error condition currently does not exist.
PAGE_TABLE_ERROR	4	RO	0b	1 = This error condition currently exists. 0 = This error condition currently does not exist.
Reserved	3:1	RO	0h	Reserved
CLAIM_ERROR	0	RO	0b	No RMbus claim occurred. Logging the error. 1 = This error condition currently exists. 0 = This error condition currently does not exist.

Power Context Base Register (PCBR)

MMIO Offset Address: 18_2120oh

Description: This provides an base address for context save/restore of GT and Media power context to DRAM.

The BIOS is expected to program this register and ensure proper allocation within Gfx stolen memory.

Field Name	Bit	Access	Reset Value	Description
Reserved	63:32	RO	000000h	Reserved
Power Context Address	31:12	RW-L	0000000h	4KB aligned address Locked with bit 0
Reserved	11:1	RO	000h	Reserved
Power Context Register Lock	0	RW-L	0b	Writing a '1' to this register locks this bit – preventing further updates. The Power Context Address bits are also locked.

GTLC: SA MMIO Registers

For Valleyview, the graphics interface will contain these registers.

This is range: 10_0000 to 17_FFFFh.

Table: GTLC SA MMIO registers

Name	Address	Functionality	Notes
Fence Registers	0x10_0000 to 0x10_007Fh	Fence Registers	Determine whether GMADR subregion is linear/TileX/TileY.
Gfx Flush Control	0x10_1008h	Invalidates TLB	Invalidates the TLB.
GTFIFOCTL	0x12_0008	GT FIFO Control	
GTLC Wake Control	0x13_0090	GTLC Wake Control	
GTLC Power Well Status	0x13_0094	GTLC Power Well Status	
Render Force Wake Req	0x13_00B0		
Render Force Wake Ack	0x13_00B4		
Media Force Wake Req	0x13_00B8		
Media Force Wake Ack	0x13_00BC		
Render RC6 (Standby) counter	0x13_8108		
Media RC6 (Standby) counter	0x13_810C		

Fence Registers

MMIO Address Offset: 10_0000h

Description: Fence Registers

Fence0 Register: 10_0000h

Fence1 Register: 10_0008h

....

Fence15 Register: 10_0078h

Field Name	Bit	Access	Reset Value	Description
FENCEUP	63:44	RW	00000000h	Bits 31:12 of the ending Graphics Address of the fence region. Fence regions must be aligned to a 4KB page. This address represents the last 4KB page of the fence region (Upper Bound is included in the fence region). Graphics Address is the offset within GMADR space.
Reserved	43	RO	0b	Reserved
Pitch	42:32	RW	000h	This field specifies the width (pitch) of the fence region in multiple of "tile widths". For Tile X this field must be programmed to a multiple of 512B ("003" is the minimum value) and for Tile Y this field must be programmed to a multiple of 128B ("000" is the minimum value). 000h = 128B 001h = 256B 002h = 384B 003h = 512B 004h = 640B 005h = 768B 006h = 896B 007h = 1024B ... 3FFh = 128KB

Field Name	Bit	Access	Reset Value	Description
				4FFh = 160KB 5FFh = 192KB 6FFh = 224KB 7FFh = 256KB
FENCELO	31:12	RW	000000h	Bits 31:12 of the starting Graphics Address of the fence region. Fence regions must be aligned to 4KB. This address represents the first 4KB page of the fence region (Lower Bound is included in the fence region). Graphics Address is the offset within GMADR.
Reserved	11:2	RO	000h	Reserved.
TILE	1	RW	0b	**This field specifies the spatial ordering of QW within tiles.** 0b - Consecutive SWords (32B) sequenced in the X direction 1b - Consecutive OWords (16B) sequenced in the Y direction
FENCEVAL	0	RW	0b	This field specifies whether or not this fence register defines a fence region. 0b - FENCE INVALID 1b - FENCE VALID

Gfx_FLSH_CNTL

MMIO Address Offset: 10_1008Description:

Description: Used to flush the TLB.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:1	RO	00000000h	Reserved
GfxFlshCntl	0	WO	0b	A write to this bit flushes the GFX TLB in the graphics interface. The data associated with the write is discarded and a read returns all 0s.

GTFIFOCTL

MMIO Address Offset: 12_0008h

Description: GT FIFO Control. Number of free Wake FIFO entries.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:7	RW	0b	Reserved (should not be changed)
GT_FIFO_FREE_ENTRIES	6:0	RO	40h	This value indicate the number of entries that are free within the GT FIFO. The free entries may be used for IA accesses and IO accesses to GT. The Wake FIFO depth is 64 entries. Zero entries free = "threshold has been hit" condition.

GTLC Wake Control

MMIO Address Offset: 13_0090h

Description: GT Wake Control. This register is used as a way to control the GTLC Render and Media Power wells.

Writing bit 0 during normal operation may result in a hang.

Field Name	Bit	Access	*Reset Value*	Description
Reserved	31:26	RO	00h	Reserved
RenderContextExists	25	RW	0b	• SW can only write a one to this bit. The usage is for support of s0ix, where the Driver would restore the value in this bit no later than the time a '1' is written to ALLOWWAKEREQ (bit 0 of this register) during s0ix exit. SW should not write '1' to this bit during a cold boot exit or warm reset exit. • CZ reset is the only thing that can clear this bit. • This bit is set by HW after the first Render context save after cold boot or warm reset. • SW can only write a one to this bit. If SW tries to write a zero, the previous value will be

Field Name	Bit	Access	*Reset Value*	Description
				maintained.
MediaContextExists	24	RW	0b	• SW can only write a one to this bit. The usage is for support of s0ix, where the Driver would restore the value in this bit no later than the time a '1' is written to ALLOWWAKEREQ (bit 0 of this register) during s0ix exit. SW should not write '1' to this bit during a cold boot exit or warm reset exit. • CZ reset is the only thing that can clear this bit. • This bit is set by HW after the first Media context save after cold boot or warm reset. • SW can only write a one to this bit. If SW tries to write a zero, the previous value will be maintained.
Reserved	23:17	RO	00h	Reserved
Reserved	16	RW	0b	Reserved
Reserved	15:9	RO	00h	Reserved
Reserved	8	RW	0b	Reserved
Reserved	7:1	RO	00h	Reserved
ALLOWWAKEREQ	0	RW	0	This bit is used as a way for the driver to make sure GTLC Render AND MEDIA engines do not wake while powered down. The usage is specifically with s0ix, where the driver wants to access the message channel common well (GSclk domain) and does not want the render/media wells to wake up. To remove ambiguity, this bit should be set to '1' and the ALLOWWAKEACK should be observed to the '1' before the FWAKEMEDIAREQ/FWAKERENDERREQ are set. Care must be taken to ONLY use this register prior to powering down the graphics interface for S0ix or after registers have been initialized.

GTLC Power Well Status

MMIO Address Offset: 13_0094h

Description: This register contains Ack and status information for power well requests.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:8	RO	000000h	Reserved
Reserved	7	RO	0b	Reserved
Reserved	6	RO	0b	Reserved
Reserved	5	RO	0b	Reserved
Reserved	4	RO	0b	Reserved
Reserved	3:2	RO	00b	Reserved
ALLOWWAKEERR	1	RW1C	0b	HW set, SW cleared. When access to media or render is observed when ALLOWWAKE=0, the ALLOWWAKERR bit will be set. It will be up to SW or a power cycle to clear the ALLOWWAKEERR bit.
ALLOWWAKEACK	0	RO	0b	Indicates that the allow wake request has been completed.

Render Force wake req

MMIO Address Offset: 13_00B0h

Description: This register contains per thread force wake request bits for the Render power Well.. The upper 16 bits act as masks for the lower 16 bits. bit-31 masks bit-15 and bit 16 masks bit 0.

1. Driver writes to GPM force wake request bit. (VV will have a Render(13_00B0(15:0)) and a Media (13_00B8(15:0)) bits)
2. The GPM responds by writing (via PLINK) to 1300B4[15:0] or 1300BC[15:0] register.
3. Driver polls (1300B4[15:0] and/or 1300BC[15:0])status until 1 ... indicating that that well has completed wake sequence.

Since the registers are per thread, only the specific bit that was forced should be checked for status.

Field Name	Bit	Access	Reset Value	Description
FWAKERENDERREQMSK	31:16	RO	0000h	Mask bits for lower 16 bits to avoid a read modify/write. If '0', the corresponding bit in [15:0] is not changed. If '1', the corresponding bit in [15:0] is changed to the value in [15:0].
FWAKERENDERREQ	15:0	RW	0000h	Force Wake Render request bits.

Render Force wake ack

MMIO Address Offset: 13_00B4h

Description: This register contains the per thread force wake acknowledge bits for the Render power well.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:16	RO	0000h	Reserved
FWAKERENDERACK	15:0	RW	0000h	Force Wake Render request bits. Driver must poll on the corresponding bit to confirm that the well has woken. For example, if 13_00B0[0] is written to a '1' (along with 13_00B0[16]='1'), then bit0 of this register indicates when the force wake request has been completed.

Media Force wake req

MMIO Address Offset: 13_00B8h

Description: This register contains per thread force wake request bits for the Media power Well. The upper 16 bits act as masks for the lower 16 bits. bit-31 masks bit-15 and bit 16 masks bit 0.

Field Name	Bit	Access	Reset Value	Description
FWAKEMEDIAREQMSK	31:16	RO	0000h	Mask bits for lower 16 bits to avoid a read modify/write. If '0', the corresponding bit in [15:0] is not changed. If '1', the corresponding bit in [15:0] is changed to the value in [15:0]
FWAKEMEDIAREQ	15:0	RW	0000h	Force Wake Media request bits

Media Force wake ack

MMIO Address Offset: 13_00BCh

Description: This register contains the per thread force wake acknowledge bits for the Media power well.

Field Name	Bit	Access	Reset Value	Description
Reserved	31:16	RO	0000h	Reserved
FWAKEMEDIAACK	15:0	RW	0000h	Force Wake Media request bits. Driver must poll on the corresponding bit to confirm that the well has woken. For example, if 13_00B0[0] is written to a '1' (along with 13_00B0[16]='1'), then bit0 of this register indicates when the force wake request has been completed.

Render RC6 (Standby) counter

MMIO Address Offset: 13_8108h

Description: This register contains the total RC6 residency (Render power gated and clock gated) time that Render power well was in since boot. The counter will wrap around. The time is given in units of 1.28 uSec.The 40-bit HW counter will wrap around. The only clear condition is reset. When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported. The units are CZ clock cycles.

It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.

0x13_8104[0] controls if this register should count or if it should be gated: 0= clear, 1=count

Field Name	Bit	Access	Reset Value	Description
RENDERRC6TIME	31:0	RO	00000000h	Render Residency Counter

Media RC6 (Standby) counter

MMIO Address Offset: 13_810Ch

Description: This register contains the total RC6 residency (Media power gated and clock gated) time that Media was in since boot. The counter will wrap around. The time is given in units of cz clock cycles.

The 40-bit HW counter will wrap around. The only clear condition is CZ reset. When 0x13_8104[15] is set to zero, bits 31:0 of the 40-bit counter are reported. When 0x13_8104[15] is set to one, bits 39:8 of the 40-bit counter are reported. The units are CZ clock cycles.

It is up to SW to periodically read this register and do a difference from the last time it was read. The primary usage for this register is for power characterization.

0x13_8104[1] controls if this register should count or if it should be gated: 0= clear, 1=count

Field Name	Bit	Access	Reset Value	Description
MEDIARC6TIME	31:0	RO	00000000h	Media Residency Counter

GTLC: "Render/Media/PM" Interrupt Architecture

GTLC can send status messages for Render/Media and for Power Management. There are separate sets of IIR/IMR/IER/ISR registers for Render/Media and for Power Management.

The graphics interface will house these registers.

Table: GTLC "Display" MMIO registers

Name	Address	Functionality	Notes
GTLC MIR	0x4_400Ch		GTLC Master Interrupt Enable
GTLC ISR	0x4_4010h		
GTLC IMR	0x4_4014h		
GTLC IIR	0x4_4018h		
GTLC IER	0x4_401Ch		
PM ISR	0x4_4020h		
PM IMR	0x4_4024h		
PM IIR	0x4_4028h		
PM IER	0x4_402Ch		
SWF	0x4_F000h to 0x4_F08Fh		Software Flag registers. Used by software
GTSCRATCH	0x4_F100h to 0x4_F11Fh		GT Scratch pad registers. Used by software.

GTLC: Master Interrupt Register

MMIO Address Offset: 4400Ch

Description: GTLC Master Interrupt Control Register.

For VLV, the display engine has its own interrupt registers. As a result, only the GTLC Master interrupt bit remains. The graphics interface will house these registers.

<table>
<tr><th colspan="2">GTLC Master Interrupt Register</th></tr>
<tr><td>Register Type:</td><td>MMIO</td></tr>
<tr><td>Project:</td><td>All</td></tr>
<tr><td>Default Value:</td><td>00000000h</td></tr>
<tr><td>Access Type:</td><td>Read Only, R/W</td></tr>
<tr><td>Size (in bits):</td><td>32</td></tr>
<tr><td colspan="2">For VV, this register simply controls GTLC (Render/Media/PM) ability to generate interrupts.</td></tr>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>31</td><td>Read-Write. GTLC Master_Interrupt_Control
<table>
<tr><td>Project:</td><td>Format:</td></tr>
<tr><td>All</td><td>MB1 for GTLC interrupts</td></tr>
<tr><td colspan="2">This is the master control for the GTLC to CPU interrupts. This bit must be set to 1 for any GTLC interrupts to propagate to the system.</td></tr>
</table></td></tr>
<tr><td>30:0</td><td>Read-only. Reserved for VV</td></tr>
</table>

GTLC: "Render/Media" Interrupts

Bit Definition for Interrupt Control Registers. All following interrupts are originated from GT in message format. DE needs to convert the message bits to events mapping one to one to following register.

Bit	Description
Blitter Interrupts (BCS bit31-bit22)	
31:30	*Reserved for Blitter Command Streamer*
29	**Blitter page directory faults:** This is a write of logic1 via interrupt message from GT via 0x50200 bit29
28	*Reserved for Blitter Command Streamer*
27	*Reserved for Blitter Command Streamer*
26	**Blitter MI_FLUSH_DW notify:** This is a write of logic1 via interrupt message from GT via 0x50200 bit26
25	**Blitter Command Streamererror interrupt:** This is a write of logic1 via interrupt message from GT via 0x50200 bit25
24	**Billter MMIO sync flush status:** This is a write of logic1 via interrupt message from GT via 0x50200 bit24
23	*Reserved for Blitter Command Streamer*
22	**Blitter Command Streamer MI_USER_INTERRUPT:** This is a write of logic1 via interrupt message from GT via 0x50200 bit22
Media Interrupts (VCS bit21-bit12)	
21	*Reserved for Video Command Streamer*
19	**Video page directory faults:** This is a write of logic1 via interrupt message from GT via 0x50200 bit19
18	**Video Command Streamer Watchdog counter exceeded:** This is a write of logic1 via interrupt message from GT via 0x50200 bit18
16	**Video MI_FLUSH_DW notify:** This is a write of logic1 via interrupt message from GT via 0x50200 bit16
15	**Video Command Streamer error interrupt:** This is a write of logic1 via interrupt message from GT via 0x50200 bit15
14	**Video MMIO sync flush status:** This is a write of logic1 via interrupt message from GT via 0x50200 bit14
12	**Video Command Streamer MI_USER_INTERRUPT:** This is a write of logic1 via interrupt message from GT via 0x50200 bit12
Render Interrupts (CS bit11-bit0)	
11:9	*Reserved for Main Command Streamer*
7	**Render page directory faults:** This is a write of logic1 via interrupt message from GT via 0x50200 bit7
6	**Render Command Streamer Watchdog counter exceeded:** This is a write of logic1 via interrupt message from GT via 0x50200 bit6
5	**L3 Parity Error Interrupt**
4	**Render PIPE_CONTROL notify:** This is a write of logic1 via interrupt message from GT via 0x50200 bit4
3	**Render Command Streamer error interrupt:** This is a write of logic1 via interrupt message from GT via 0x50200 bit3
2	**Render MMIO sync flush status:** This is a write of logic1 via interrupt message from GT via 0x50200 bit2
1	
0	**Render Command Streamer MI_USER_INTERRUPT:** This is a write of logic1 via interrupt message from GT via 0x50200 bit0

ISR – Interrupt Status Register

MMIO Address Offset: 4_4010h

Description: Interrupt Status

ISR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	Read Only
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Status_Bits** Project: All This field contains the non-persistent values of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR.

Value	Name	Description	Project
0b	Condition Doesn't exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes

Some inputs to this register are short pulses; therefore software should not expect to use this register to sample these conditions.

IMR – Interrupt Mask Register

MMIO Address Offset: 4_4014h

Description: Interrupt Mask Register

IMR	
Register Type:	MMIO
Project:	All
Default Value:	FFFFFFFFh
Access Type:	R/W
Size (in bits):	32

For GT command streamer interrupts DO NOT use this register to mask interrupt events. Instead use the individual GT command streamer MASK bits.
For PM interrupts DO NOT use this register to mask interrupt events. Instead use the individual PM MASK bits in the corresponding PMunit register space.
See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Mask_Bits** Project: All This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

IIR

MMIO Address Offset: 4_4018h

Description: Interrupt Identity Register

IIR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W Clear
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Identity_Bits** Project: All This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits. For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit will momentarily go low, then return high to indicate there is another interrupt pending

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected (may or may not have generated a CPU interrupt)	All

IER – Interrupt Enable Register

MMIO Address Offset: 4_401Ch

Description: Interrupt Enable Register

IER	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Enable_Bits** Project: All The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the IIR becomes set. A disabled interrupt will still appear in the IIR register to allow polling of interrupt sources. The DE_IER master interrupt control bit must be set to 1 for any interrupts to propagate to the system.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

GTLC: "Power Management" Interrupts

The graphics interface routes the GTLC PM writes to the appropriate ISR/IIR registers. The bit assignments are below.

Power Management Interrupt Bit Definition	
Project:	All
Size (in bits):	32
Power Management interrupt bits come to the graphics interface via direct wires. The GT_IIR and PM_IIR are ORed together to generate the CPU interrupt. The Power Management Interrupt Control Registers all share the same bit definitions from this table.	

Bit	Description
31:7	**Reserved** Project: All; Format: MBZ
6	**Render Frequency Downward Timeout During RC6 interrupt:** This is a write of logic1 via interrupt message from GT via 0x50210 bit6
5	**RP UP threshold interrupt:** This is a write of logic1 via interrupt message from GT via 0x50210 bit5
4	**RP DOWN threshold interrupt:** This is a write of logic1 via interrupt message from GT via 0x50210 bit4
3	**Reserved** Project: All; Format: MBZ
2	**Render geyserville UP evaluation interval interrupt:** This is a write of logic1 via interrupt message from GT via 0x50210 bit2
1	**Render geyserville Down evaluation interval interrupt:** This is a write of logic1 via interrupt message from GT via 0x50210 bit1
0	*Reserved for GT PM unit*

ISR – Interrupt Status Register

MMIO Address Offset: 4_4020h

Description: Interrupt Status Register

ISR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	Read Only
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Status_Bits** Project: All This field contains the non-persistent values of all interrupt status bits. The IMR register selects which of these interrupt conditions are reported in the persistent IIR.

Value	Name	Description	Project
0b	Condition Doesn't exist	Interrupt Condition currently does not exist	All
1b	Condition Exists	Interrupt Condition currently exists	All

Programming Notes

Some inputs to this register are short pulses; therefore software should not expect to use this register to sample these conditions.

IMR – Interrupt Mask Register

MMIO Address Offset: 4_4024h

Description: Interrupt Mask Register

IMR	
Register Type:	MMIO
Project:	All
Default Value:	FFFFFFFFh
Access Type:	R/W
Size (in bits):	32

For GT command streamer interrupts DO NOT use this register to mask interrupt events. Instead use the individual GT command streamer MASK bits.
For PM interrupts DO NOT use this register to mask interrupt events. Instead use the individual PM MASK bits in the corresponding PMunit register space.
See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Mask_Bits** Project: All This field contains a bit mask which selects which interrupt bits from the ISR are reported in the IIR.

Value	Name	Description	Project
0b	Not Masked	Not Masked – will be reported in the IIR	All
1b	Masked	Masked – will not be reported in the IIR	All

IIR – Interrupt Identity Register

MMIO Address Offset: 4_4028h

Description: Interrupt Identity Register

IIR	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W Clear
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Identity_Bits** Project: All This field holds the persistent values of the interrupt bits from the ISR which are unmasked by the IMR. If enabled by the IER, bits set in this register will generate a CPU interrupt. Bits set in this register will remain set (persist) until the interrupt condition is cleared by writing a '1' to the appropriate bits. For each bit, the IIR can store a second pending interrupt if two or more of the same interrupt conditions occur before the first condition is cleared. Upon clearing the interrupt, the IIR bit will momentarily go low, then return high to indicate there is another interrupt pending

Value	Name	Description	Project
0b	Condition Not Detected	Interrupt Condition Not Detected	All
1b	Condition Detected	Interrupt Condition Detected (may or may not have generated a CPU interrupt)	All

IER – Interrupt Enable Register

MMIO Address Offset: 4_402Ch

Description: Interrupt Enable Register

IER	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W
Size (in bits):	32

See the interrupt bit definition tables to find the source event for each interrupt bit.

Bit	Description
31:0	**Interrupt_Enable_Bits** Project: All The bits in this register enable a CPU interrupt to be generated whenever the corresponding bit in the IIR becomes set. A disabled interrupt will still appear in the IIR register to allow polling of interrupt sources. The DE_IER master interrupt control bit must be set to 1 for any interrupts to propagate to the system.

Value	Name	Description	Project
0b	Disable	Disable	All
1b	Enable	Enable	All

GTLC: Scratch Registers

The GTLC engine can target these registers with information. The driver is then able to read the registers.

SWF – Software Flags

MMIO Address Offset: 0x4_F000 – 0x4_F08C

Description: These (36 Dword) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these registers is defined by the software architecture.

SWF	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W
Size (in bits):	36x32

These registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these registers is defined by the software architecture.

Dword	Bit	Description
0..35	31:0	**Software_Flags** Project: All

GTSCRATCH – GT Scratch registers

MMIO Address Offset: 0x4_F100 – 0x4_F11F

Description: These (8 Dword) registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these registers is defined by the software architecture.

GTSCRATCH	
Register Type:	MMIO
Project:	All
Default Value:	00000000h
Access Type:	R/W
Size (in bits):	8x32

These registers are used as scratch pad data storage space and have no direct effect on hardware operation. The use of these registers is defined by the software architecture.

Dword	Bit	Description
0..7	31:0	**GT_Scratchpad** Project: All

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 3: GPU Overview

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Introduction **1**
Graphics Processing Unit (GPU) 1
GPU Overview 2
Command Stream (CS) Unit 3
3D Pipeline 3
Media Pipeline 3
Thread Dispatching 3
Execution Units (EUs) 3
Shared Functions 4
Fixed and Shared Function IDs 5
Video Codec Engine 6
Register Address Maps **7**
Graphics Register Address Map 7
VGA and Extended VGA Register Map 7
Memory Object Overview **12**
Hardware Status Page 13
Memory Access Indirection 13
Instruction Ring Buffers 16
Instruction Batch Buffers 16
Logical Contexts 17
Memory Data Formats **26**
Unsigned Normalized (UNORM) 26
Gamma Conversion (SRGB) 26
Signed Normalized (SNORM) 26
Unsigned Integer (UINT/USCALED) 27
Signed Integer (SINT/SSCALED) 27
Floating Point (FLOAT) 27

Introduction

The integrated graphics component, specifically called the Graphics Processing Unit, or GPU, resides on the same chip die as the Central Processing Unit, or CPU, and communicates with the CPU via the on-chip bus, with internal memory and with output device(s). As Intel GPUs have evolved, they now occupy a significant percentage of space on the chip, and provide customers with high performance and low-power graphics processing, eliminating the need to purchase a separate video card for most users.

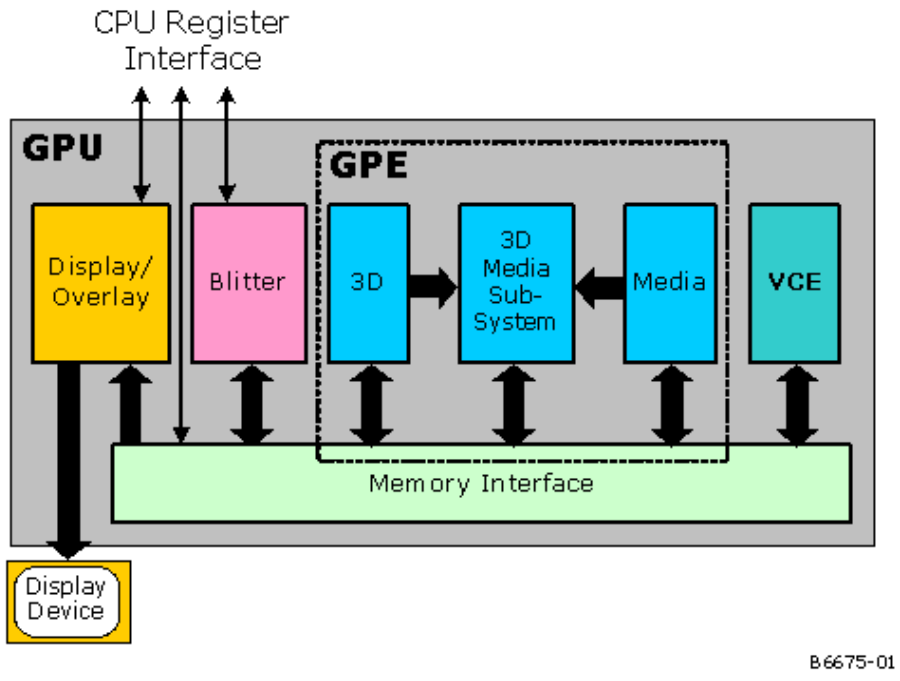
This Behavioral Specification, or BSpec for short, provides detailed narrative and referential information required by graphics device driver engineers and graphics API-level programmers to take advantage of the sophisticated architecture and programmability of the GPU.

Graphics Processing Unit (GPU)

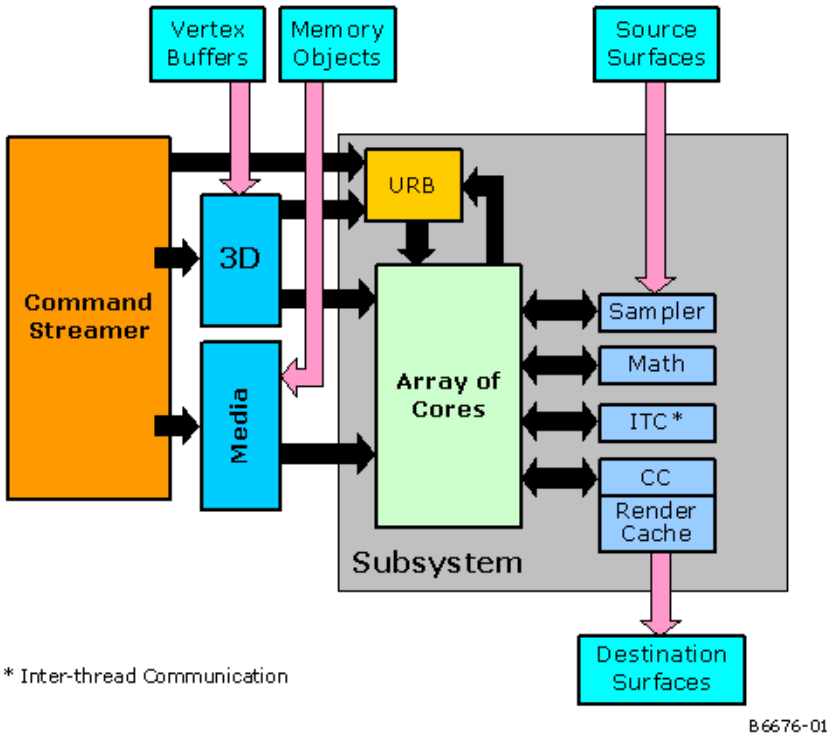
The Graphics Processing Unit is controlled by the CPU through a direct interface of memory-mapped IO registers, and indirectly by parsing commands that the CPU has placed in memory. The Display interface and Blitter (**bl**ock **i**mage **t**ransfer**er**) are controlled primarily by direct CPU register addresses, while the 3D and Media pipelines and the parallel Video Codec Engine (VCE) are controlled primarily through instruction lists in memory.

The subsystem contains an array of cores, or execution units, with a number of "shared functions", which receive and process messages at the request of programs running on the cores. The shared functions perform critical tasks, such as sampling textures and updating the render target (usually the frame buffer). The cores themselves are described by an instruction set architecture, or ISA.

Block Diagram of the GPU



GPU Overview



The subsystem consists of an array of *execution units* (*EUs*, sometimes referred to as an array of *cores*) along with a set of *shared functions* outside the EUs that the EUs leverage for I/O and for complex computations. Programmers access the subsystem via the 3D or Media pipelines.

EUs are general-purpose programmable cores that support a rich instruction set that has been optimized to support various 3D API shader languages as well as media functions (primarily video) processing.

Shared functions are hardware units which serve to provide specialized supplemental functionality for the EUs. A shared function is implemented where the demand for a given specialized function is insufficient to justify the costs on a per-EU basis. Instead a single instantiation of that specialized function is implemented as a stand-alone entity outside the EUs and shared among the EUs.

Invocation of the shared functionality is performed via a communication mechanism called a *message*. A message is a small self-contained packet of information created by a kernel and directed to a specific shared function. For SNB, the message is defined by a range of MRF registers that hold message operands, a destination shared function ID, a function-specific encoding of the desired operation, and a destination GRF register to which any writeback response is to be directed. Messages are dispatched to the shared function under software control via the send instruction. This instruction identifies the contents of the message and the GRF register locations to direct any response.

The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

Command Stream (CS) Unit

The Command Stream (CS) unit manages the use of the 3D and Media pipelines; it performs switching between pipelines and forwarding command streams to the currently active pipeline. It manages allocation of the URB and helps support the Constant URB Entry (CURBE) function.

3D Pipeline

The 3D Pipeline provides specialized 3D primitive processing functions. These functions are provided by a pipeline of "fixed function" stages (units) and GEN threads spawned by these units. See *3D Pipeline Overview*.

Media Pipeline

The Media pipeline provides both specialized media-related processing functions and the ability to perform more general ("generic") functionality. These Media-specific functions are provided by a Video Front End (VFE) unit. A Thread Spawner (TS) unit is utilized to spawn GEN threads requested by the VFE unit, or as required when the pipeline is used for general processing. See *Media Pipeline Overview*.

Thread Dispatching

When the 3D and Media pipelines send requests for thread initiation to the Subsystem, the thread Dispatcher receives the requests. The dispatcher performs such tasks as arbitrating between concurrent requests, assigning requested threads to hardware threads on EUs, allocating register space in each EU among multiple threads, and initializing a thread's registers with data from the fixed functions and from the URB. This operation is largely transparent to software.

Execution Units (EUs)

The Execution Units (EUs) are the programmable shader units of the Gen Architecture. Each is a stand-alone programmable computational unit used for execution of 3D shaders and media/gpgpu kernels. Internally each is capable of multi-issue SIMD execution, and their hardware multi-threaded operation provides a very high-efficiency execution environment in the face of long data latencies typically associated with memory accesses. Each hardware thread within and EU has a dedicated large-capacity high-bandwidth register file (GRF) and associated independent thread-state . Execution is multi-issue per clock to pipelines capable of integer, single and double precision floating point operations, SIMD branch capability, logical operations, transcendental operations, and other miscellaneous operations. Communication to support units (shared functions) for operations such as texture sampling or scatter/gather load/stores is via 'messages' programmatically constructed and 'sent' to those functions, with dependency hardware causing the issuing thread to sleep until the requested data has been returned.

EU instance count varies by product generation, as well as by SKU within a given generation, and their capabilities have evolved over the many generation of the Gen Architecture.. Please see "Device Attributes" in the "Configuration" chapter for specific rates and capacities associated with Execution Units.

Shared Functions

Shared functions are hardware units that provide specialized supplemental functionality for the EUs. A shared function is implemented where the demand for a given specialized function is insufficient to justify the costs on a per-EU basis. Instead a single instantiation of that specialized function is implemented as a stand-alone entity outside the EUs and shared among the EUs.

Invocation of the shared functionality is performed via a communication mechanism called a message. A message is a small self-contained packet of information created by a kernel and directed to a specific shared function.

Programming Note	
Context:	Communication mechanism in shared functions
The message is defined by a range of Message Register File (MRF) registers that hold message operands, a destination shared function ID, a function-specific encoding of the desired operation, and a destination General Register File (GRF) register to which any writeback response is directed.	

Messages are dispatched to the shared function under software control via the *send* instruction. This instruction identifies the contents of the message and the GRF register locations to direct any response.

The message construction and delivery mechanisms are general in their definition and capable of supporting a wide variety of shared functions.

Fixed and Shared Function IDs

The following table lists the assignments (encodings) of the Shared Function and Fixed Function IDs used within the GPE. A Shared Function is a valid target of a message initiated via a *send* instruction. A Fixed Function is an identifiable unit of the 3D or Media pipeline. Note that the Thread Spawner is both a Shared Function and Fixed Function.

Function IDs

ID[3:0]	SFID	Shared Function	FFID	Fixed Function
0x0	SFID_NULL	Null	FFID_NULL	Null
0x1	Reserved	---	Reserved	---
0x2	SFID_SAMPLER	Sampler	Reserved	---
0x3	SFID_GATEWAY	Message Gateway	Reserved	---
0x4	SFID_DP_SAMPLER	Sampler Cache Data Port	FFID_HS	Hull Shader
0x5	SFID_DP_RC	Render Cache Data Port	FFID_DS	Domain Shader
0x6	SFID_URB	URB	Reserved	---
0x7	SFID_SPAWNER	Thread Spawner	FFID_SPAWNER	Thread Spawner
0x8	SFID_VME	Video Motion Estimation	Reserved	---
0x9	SFID_DP_CC	Constant Cache Data Port	FFID_VS	Vertex Shader
0xA	SFID_DP_DC0	Data Cache Data Port0	FFID_CS	Command Stream
0xB	SFID_PI	Pixel Interpolator	FFID_VF	Vertex Fetch
0xC	SFID_DP_DC1	Data Cache Data Port1	FFID_GS	Geometry Shader
0xD	SFID_CRE	Check & Refinement Engine	Reserved	---
0xE	Reserved	---	FFID_SF	Strip/Fan Unit
0xF	Reserved	---	FFID_WM	Windower/Masker Unit

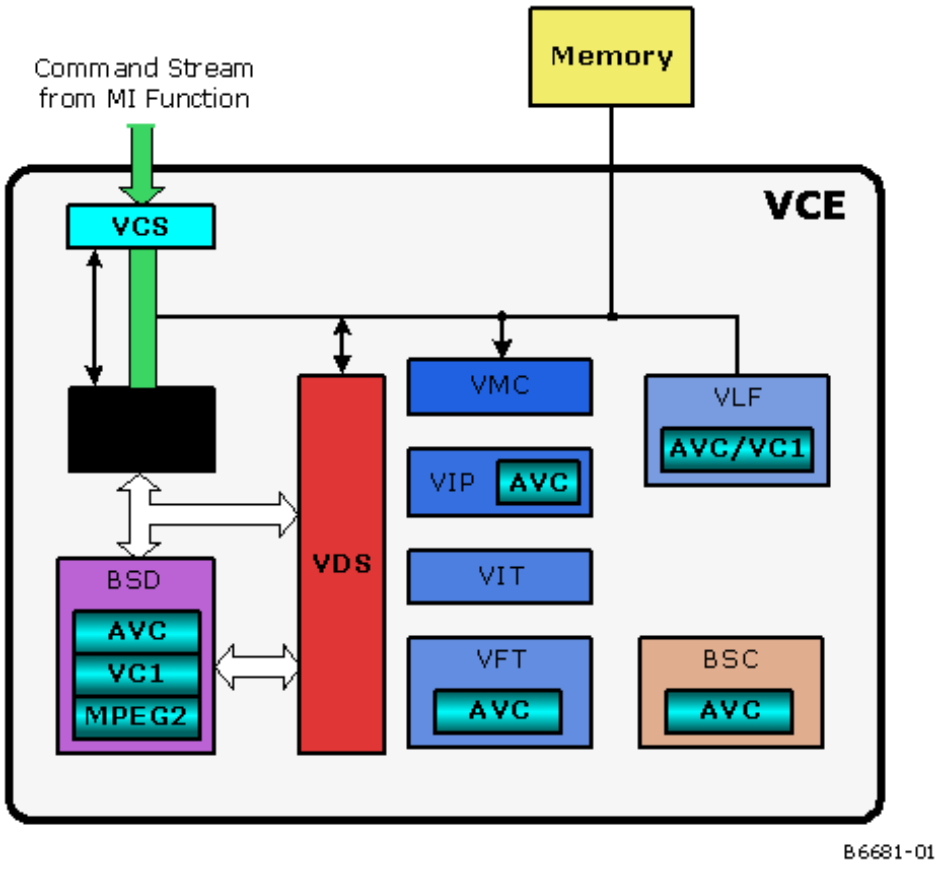
Programming Note	
Context:	Shared Function ID
SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for for more messages types. They act as a single logical entity.	

Video Codec Engine

The parallel Video Codec Engine (VCE) is a fixed function video decoder and encoder engine. It is also referred to as the multi-format codec (MFX) engine, as a unified fixed function pipeline is implemented to support multiple video coding standards such as MPEG2, VC1, and AVC:

- VCS – VCE Command Streamer unit (also referred to as BCS)
- BSD – Bitstream Decoder unit
- VDS – Video Dispatcher unit
- VMC – Video Motion Compensation unit
- VIP – Video Intra Prediction unit
- VIT – Video Inverse Transform unit
- VLF – Video Loop Filter unit
- VFT – Video Forward Transform unit (encoder only)
- BSC – Bitstream Encoder unit (encoder only)

VCE Diagram



Device	AVC BSD	VC1 BSD	AVC Dec	VC1 Dec	MPEG2 Dec	AVC Enc
	No	No	Yes	Yes	Yes	Yes

Register Address Maps

Graphics Register Address Map

This chapter provides address maps of the graphics controllers I/O and memory-mapped registers. Individual register bit field descriptions are provided in the following chapters. PCI configuration address maps and register bit descriptions are provided in the following chapter.

Memory and IO Space Registers

These are graphics MMIO ranges used for [CHV]. Note that this is only a subset of the complete definition of the MMIO address space.

Range Start (Hex)	Range End (Hex)	Unit Owning the Range
00002000	00002FFF	Render/Generic Media Engine
00004000	00004FFF	Render/Generic Media Graphics Memory Arbiter
00012000	000123FF	MFX Control Engine (Video Command Streamer)
00012400	00012FFF	Media Units (VIN Unit)
00014000	00014FFF	MFX Memory Arbiter
00022000	00022FFF	Blitter Engine
00024000	00024FFF	Blitter Memory Arbiter
00100000	00107FFF	Fence Registers
00140000	0017FFFF	MCHBAR (SA)

Programming Note	
Context:	Memory and IO space registers, graphics MMIO ranges
8800h-88FFh is a reserved range for GT. IA accesses to this region have no impact.	

Programming Note	
Context:	Memory and IO space registers, graphics MMIO ranges
TAP (backdoor) accesses to the reserved range of 8800h-88FFh result in hardware hangs. Do not use it.	

VGA and Extended VGA Register Map

For I/O locations, the value in the address column represents the register I/O address. For memory mapped locations, this address is an offset from the base address programmed in the MMADR register.

VGA and Extended VGA I/O and Memory Register Map

Address	Register Name (Read)	Register Name (Write)
2D Registers		
3B0h–3B3h	Reserved	Reserved
3B4h	VGA CRTC Index (CRX) (monochrome)	VGA CRTC Index (CRX) (monochrome)
3B5h	VGA CRTC Data (monochrome)	VGA CRTC Data (monochrome)
3B6h–3B9h	Reserved	Reserved
3Bah	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3BBh–3BFh	Reserved	Reserved
3C0h	VGA Attribute Controller Index (ARX)	VGA Attribute Controller Index (ARX)/ VGA Attribute Controller Data (alternating writes select ARX or write ARxx Data)
3C1h	VGA Attribute Controller Data (read ARxx data)	Reserved
3C2h	VGA Feature Read Register (ST00)	VGA Miscellaneous Output Register (MSR)
3C3h	Reserved	Reserved
3C4h	VGA Sequencer Index (SRX)	VGA Sequencer Index (SRX)
3C5h	VGA Sequencer Data (SRxx)	VGA Sequencer Data (SRxx)
3C6h	VGA Color Palette Mask (DACMASK)	VGA Color Palette Mask (DACMASK)
3C7h	VGA Color Palette State (DACSTATE)	VGA Color Palette Read Mode Index (DACRX)
3C8h	VGA Color Palette Write Mode Index (DACWX)	VGA Color Palette Write Mode Index (DACWX)
3C9h	VGA Color Palette Data (DACDATA)	VGA Color Palette Data (DACDATA)
3CAh	VGA Feature Control Register (FCR)	Reserved
3CBh	Reserved	Reserved
3CCh	VGA Miscellaneous Output Register (MSR)	Reserved
3CDh	Reserved	Reserved
3CEh	VGA Graphics Controller Index (GRX)	VGA Graphics Controller Index (GRX)
3CFh	VGA Graphics Controller Data (GRxx)	VGA Graphics Controller Data (GRxx)
3D0h–3D1h	Reserved	Reserved
2D Registers		
3D4h	VGA CRTC Index (CRX)	VGA CRTC Index (CRX)
3D5h	VGA CRTC Data (CRxx)	VGA CRTC Data (CRxx)
System Configuration Registers		
3D6h	GFX/2D Configurations Extensions Index (XRX)	GFX/2D Configurations Extensions Index (XRX)
3D7h	GFX/2D Configurations Extensions Data (XRxx)	GFX/2D Configurations Extensions Data (XRxx)

Address	Register Name (Read)	Register Name (Write)
2D Registers		
3D8h–3D9h	Reserved	Reserved
3DAh	VGA Status Register (ST01)	VGA Feature Control Register (FCR)
3DBh–3DFh	Reserved	Reserved

Indirect VGA and Extended VGA Register Indices

The registers listed in this section are indirectly accessed by programming an index value into the appropriate SRX, GRX, ARX, or CRX register. The index and data register address locations are listed in the previous section. Additional details concerning the indirect access mechanism are provided in the *VGA and Extended VGA Register Description* Chapter (see SRxx, GRxx, ARxx or CRxx sections).

2D Sequence Registers (3C4h / 3C5h)

Index	Sym	Description
00h	SR00	Sequencer Reset
01h	SR01	Clocking Mode
02h	SR02	Plane / Map Mask
03h	SR03	Character Font
04h	SR04	Memory Mode
07h	SR07	Horizontal Character Counter Reset

2D Graphics Controller Registers (3CEh / 3CFh)

Index	Sym	Register Name
00h	GR00	Set / Reset
01h	GR01	Enable Set / Reset
02h	GR02	Color Compare
03h	GR03	Data Rotate
04h	GR04	Read Plane Select
05h	GR05	Graphics Mode
06h	GR06	Miscellaneous
07h	GR07	Color Don’t Care
08h	GR08	Bit Mask
10h	GR10	Address Mapping
11h	GR11	Page Selector
18h	GR18	Software Flags

2D Attribute Controller Registers (3C0h / 3C1h)

Index	Sym	Register Name
00h	AR00	Palette Register 0
01h	AR01	Palette Register 1
02h	AR02	Palette Register 2
03h	AR03	Palette Register 3
04h	AR04	Palette Register 4
05h	AR05	Palette Register 5
06h	AR06	Palette Register 6
07h	AR07	Palette Register 7
08h	AR08	Palette Register 8
09h	AR09	Palette Register 9
0Ah	AR0A	Palette Register A
0Bh	AR0B	Palette Register B
0Ch	AR0C	Palette Register C
0Dh	AR0D	Palette Register D
0Eh	AR0E	Palette Register E
0Fh	AR0F	Palette Register F
10h	AR10	Mode Control
11h	AR11	Overscan Color
12h	AR12	Memory Plane Enable
13h	AR13	Horizontal Pixel Panning
14h	AR14	Color Select

2D CRT Controller Registers (3B4h / 3D4h / 3B5h / 3D5h)

Index	Sym	Register Name
00h	CR00	Horizontal Total
01h	CR01	Horizontal Display Enable End
02h	CR02	Horizontal Blanking Start
03h	CR03	Horizontal Blanking End
04h	CR04	Horizontal Sync Start
05h	CR05	Horizontal Sync End
06h	CR06	Vertical Total
07h	CR07	Overflow
08h	CR08	Preset Row Scan
09h	CR09	Maximum Scan Line
0Ah	CR0A	Text Cursor Start
0Bh	CR0B	Text Cursor End
0Ch	CR0C	Start Address High
0Dh	CR0D	Start Address Low
0Eh	CR0E	Text Cursor Location High
0Fh	CR0F	Text Cursor Location Low
10h	CR10	Vertical Sync Start
11h	CR11	Vertical Sync End
12h	CR12	Vertical Display Enable End
13h	CR13	Offset
14h	CR14	Underline Location
15h	CR15	Vertical Blanking Start
16h	CR16	Vertical Blanking End
17h	CR17	CRT Mode
18h	CR18	Line Compare
22h	CR22	Memory Read Latch Data

Memory Object Overview

Any memory data accessed by the device is considered part of a *memory object* of some memory object type.

The following table lists the various memory objects types and an indication of their role in the system.

Memory Object Type	Role
Graphics Translation Table (GTT)	Contains PTEs used to translate "graphics addresses" into physical memory addresses.
Hardware Status Page	Cached page of sysmem used to provide fast driver synchronization.
Logical Context Buffer	Memory areas used to store (save/restore) images of hardware rendering contexts. Logical contexts are referenced via a pointer to the corresponding Logical Context Buffer.
Ring Buffers	Buffers used to transfer (DMA) instruction data to the device. Primary means of controlling rendering operations.
Batch Buffers	Buffers of instructions invoked indirectly from Ring Buffers.
State Descriptors	Contains state information in a prescribed layout format to be read by hardware. Many different state descriptor formats are supported.
Vertex Buffers	Buffers of 3D vertex data indirectly referenced through "indexed" 3D primitive instructions.
VGA Buffer (Must be mapped UC on PCI)	Graphics memory buffer used to drive the display output while in legacy VGA mode.
Display Surface	Memory buffer used to display images on display devices.
Overlay Surface	Memory buffer used to display overlaid images on display devices.
Overlay Register, Filter Coefficients	Memory area used to provide double-buffer for Overlay register and filter coefficient loading.
Cursor Surface	Hardware cursor pattern in memory.
2D Render Source	Surface used as primary input to 2D rendering operations.
2D Render R-M-W Destination	2D rendering output surface that is read in order to be combined in the rendering function. Destination surfaces that accessed via this Read-Modify-Write mode have somewhat different restrictions than Write-Only Destination surfaces.
2D Render Write-Only Destination	2D rendering output surface that is written but not read by the 2D rendering function. Destination surfaces that accessed via a Write-Only mode have somewhat different restrictions than Read-Modify-Write Destination surfaces.
2D Monochrome Source	1 bpp surfaces used as inputs to 2D rendering after being converted to foreground/background colors.
2D Color Pattern	8x8 pixel array used to supply the "pattern" input to 2D rendering functions.
DIB	"Device Independent Bitmap" surface containing "logical" pixel values that are converted (via LUTs) to physical colors.
3D Color Buffer	Surface receiving color output of 3D rendering operations. May also be accessed via R-M-W (aka blending). Also referred to as a Render Target.

Memory Object Type	Role
3D Depth Buffer	Surface used to hold per-pixel depth and stencil values used in 3D rendering operations. Accessed via RMW.
3D Texture Map	Color surface (or collection of surfaces) which provide texture data in 3D rendering operations.
"Non-3D" Texture	Surface read by Texture Samplers, though not in normal 3D rendering operations (for example, in video color conversion functions).
Motion Comp Surfaces	These are the Motion Comp reference pictures.
Motion Comp Correction Data Buffer	This is Motion Comp intra-coded or inter-coded correction data.

Hardware Status Page

The hardware status page is a naturally-aligned 4KB page residing in snooped system memory. This page exists primarily to allow the device to report status via PCI master writes – thereby allowing the driver to read/poll WB memory instead of UC reads of device registers or UC memory.

The address of this page is programmed via the HWS_PGA MI register. The definition of that register (in *Memory Interface Registers*) includes a description of the layout of the Hardware Status Page.

Memory Access Indirection

The GPE supports the indirection of certain graphics (GTT-mapped) memory accesses. This support comes in the form of two *base address* state variables used in certain memory address computations with the GPE.

The intent of this functionality is to support the dynamic relocation of certain driver-generated memory structures after command buffers have been generated but prior to the their submittal for execution. For example, as the driver builds the command stream it could append pipeline state descriptors, kernel binaries, etc. to a general state buffer. References to the individual items would be inserted in the command buffers as offsets from the base address of the state buffer. The state buffer could then be freely relocated prior to command buffer execution, with the driver only needing to specify the final base address of the state buffer. Two base addresses are provided to permit surface-related state (binding tables, surface state tables) to be maintained in a state buffer separate from the general state buffer.

While the use of these base addresses is unconditional, the indirection can be effectively disabled by setting the base addresses to zero. The following table lists the various GPE memory access paths and which base address (if any) is relevant.

Base Address Utilization

Base Address Used	Memory Accesses
General State Base Address	**DataPort Read/Write DataPort memory accesses resulting from 'stateless' DataPort Read/Write requests. See** *DataPort* **for a definition of the 'stateless' form of requests.**
Dynamic State Base Address	Sampler reads of SAMPLER_STATE data and associated SAMPLER_BORDER_COLOR_STATE.
	Viewport states used by CLIP, SF, and WM/CC
	COLOR_CALC_STATE, DEPTH_STENCIL_STATE, and BLEND_STATE
	Push Constants (depeding on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)
Instruction Base Address [CHV]	**Normal EU instruction stream** (non-system routine)
	System routine EU instruction stream (starting address = SIP)
Surface State Base Address	Sampler and DataPort reads of BINDING_TABLE_STATE, as referenced by BT pointers passed via 3DSTATE_BINDING_TABLE_POINTERS
	Sampler and DataPort reads of SURFACE_STATE data
Indirect Object Base Address	**MEDIA_OBJECT Indirect Data** accessed by the CS unit .
None	CS unit reads from **Ring Buffers**, **Batch Buffers**
	CS writes resulting from PIPE_CONTROL command
	All VF unit memory accesses (**Index Buffers**, **Vertex Buffers**)
	All Sampler **Surface Memory Data** accesses (texture fetch, etc.)
	All **DataPort memory accesses**except 'stateless' DataPort Read/Write requests (e.g., RT accesses.) See *DataPort* for a definition of the 'stateless' form of requests.
	Memory reads resulting from **STATE_PREFETCH** commands
	Any **physical memory access** by the device
	GTT-mapped accesses not included above (i.e., default)
	[CHV]: Push Constants (depeding on state of **INSTPM<CONSTANT_BUFFER Address Offset Disable>**)

The following notation is used in the BSpec to distinguish between addresses and offsets:

Notation	Definition
PhysicalAddress[n:m]	Corresponding bits of a physical graphics memory byte address (not mapped by a GTT)
GraphicsAddress[n:m]	Corresponding bits of an absolute, virtual graphics memory byte address (mapped by a GTT)
GeneralStateOffset[n:m]	Corresponding bits of a relative byte offset added to the General State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
DynamicStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Dynamic State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
InstructionBaseOffset[n:m]	Corresponding bits of a relative byte offset added to the Instruction Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)
SurfaceStateOffset[n:m]	Corresponding bits of a relative byte offset added to the Surface State Base Address value, the result of which is interpreted as a virtual graphics memory byte address (mapped by a GTT)

Instruction Ring Buffers

Instruction ring buffers are the memory areas used to pass instructions to the device. Refer to the Programming Interface chapter for a description of how these buffers are used to transport instructions.

The RINGBUF register sets (defined in Memory Interface Registers) are used to specify the ring buffer memory areas. The ring buffer must start on a 4KB boundary and be allocated in linear memory. The length of any one ring buffer is limited to 2MB.

Programming Note	
Context:	Instruction Ring Buffers in memory areas.
"Indirect" 3D primitive instructions (those that access vertex buffers) must reside in the same memory space as the vertex buffers.	

Instruction Batch Buffers

Instruction batch buffers are contiguous streams of instructions referenced via an MI_BATCH_BUFFER_START and related instructions (see Memory Interface Instructions, Programming Interface). They are used to transport instructions external to ring buffers.

Programming Note	
Context:	Instruction batch buffers in memory objects
Batch buffers should not be mapped to snooped SM (PCI) addresses. The device will treat these as MainMemory (MM) addresses and, therefore, not snoop the CPU cache.	

Programming Note	
Context:	Instruction batch buffers in memory objects
The batch buffer must be QWord aligned and a multiple of QWords in length. The ending address is the address of the last valid QWord in the buffer. The length of any single batch buffer is "virtually unlimited" (i.e., could theoretically be 4GB in length).	

Logical Contexts

This section is the lead section for the following subsections:

- BSD Logical Render Context Address (LRCA)
- Video Enhancement Logical Context Data
- Logical Contexts

BSD Logical Render Context Address (LRCA)

This section discusses the following topics for the BSD Logical Render Context Address (LRCA):

- Overall Context Layout
- Register State Context
- Ring Buffer [CHV]
- Ring Context [CHV]
- The Per-Process Hardware Status Page

Overall Context Layout

Context Layout

For [CHV], when Execlists are enabled, the Context Image for the media engine consists of two 4K pages:

- Per-Process HW Status Page (4K)
- Register State Context

When Execlists are disabled (ring buffer mode of scheduling) there is no concept of context for workloads submitted to the media engine, and hence there is no context image.

Register State context is explained in detail in the “Register State Context” Section.

Register/State Context

Note: **The Register/State Context is valid only when Execlists and PPGTT are enabled.**

DW Range	DW Count	State Field	Restore Inhibited	PPGTT and Execlists Enabled	PPGTT and Execlists Disabled	Power Context	Set Before Submitting Context?
00h	1	Context Control	R	S/R	X	S/R	Yes
01h	1	Ring Head Pointer Register	R	S/R	X	S/R	Yes
02h	1	Ring Tail Pointer Register	R	R	X	S/R	Yes
03h	1	Batch Buffer Current Head Register	NR	S/R	X	S/R	No
04h	1	Batch Buffer State Register	NR	S/R	X	S/R	No
05h	1	PPGTT Directory Cache Valid Register (Software always populates via host)	R	R	X	S/R	Yes
06h	1	Reserved	X	X	X	S/R	X
07h	1	PD Base Virtual Address Register	R	R	X	S/R	Yes
08h	1	MFX_STATE_POINTER 0	NR	S/R	X	S/R	Yes
09h	1	MFX_STATE_POINTER 1	NR	S/R	X	S/R	Yes
0Ah	1	MFX_STATE_POINTER 2	NR	S/R	X	S/R	Yes
0Bh	1	MFX_STATE_POINTER 3	NR	S/R	X	S/R	Yes
0Ch	1	VCS_CNTR — Media Watchdog Counter Control	NR	S/R	X	S/R	No
0Dh	1	VCS_THRSH — Media Watchdog Counter Threshold	NR	S/R	X	S/R	No
0Eh	1	Current Context ID Register	NR	S/R	X	S/R	No
0Fh	1	Reserved	X	X	X	S/R	X

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in the Execution List mode of scheduling.

Programming Note	
Context:	Ring buffers - ring context area of LRCA
Ring Buffer registers are directly programmed in the Ring Buffer mode of scheduling.	

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information, etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution.
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

Description	MMIO Offset/Command		# of DW
NOOP		VCS	1
Load_Register_Immediate header	0x1100_1015	VCS	1
Context Control	0x12244	VCS	2
Ring Head Pointer Register	0x12034	VCS	2
Ring Tail Pointer Register	0x12030	VCS	2
RING_BUFFER_START	0x12038	VCS	2
RING_BUFFER_CONTROL	0x1203C	VCS	2
Batch Buffer Current Head Register (UDW)	0x12168	VCS	2
Batch Buffer Current Head Register	0x12140	VCS	2
Batch Buffer State Register	0x12110	VCS	2
SECOND_BB_ADDR_UDW	0x1211C	VCS	2
SECOND_BB_ADDR	0x12114	VCS	2
SECOND_BB_STATE	0x12118	VCS	2
NOOP		VCS	8
NOOP		VCS	1
Load_Register_Immediate header	0x1100_1011	VCS	1
CTX_TIMESTAMP	0x123A8	VCS	2
PDP3_UDW	0x1228C	VCS	2
PDP3_LDW	0x 12288	VCS	2

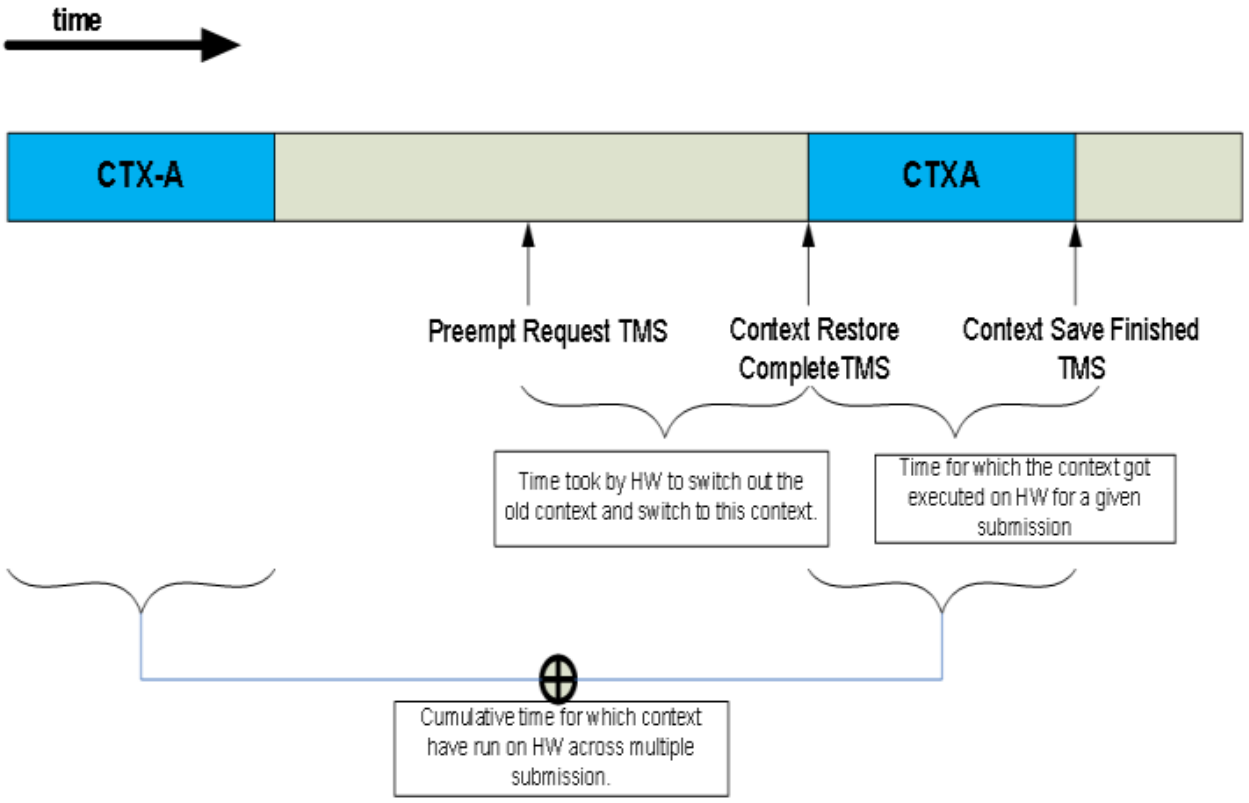
Description	MMIO Offset/Command		# of DW
PDP2_UDW	0x 12284	VCS	2
PDP2_LDW	0x 12280	VCS	2
PDP1_UDW	0x 1227C	VCS	2
PDP1_LDW	0x 12278	VCS	2
PDP0_UDW	0x 12274	VCS	2
PDP0_LDW	0x 12270	VCS	2
NOOP		VCS	12
NOOP		VCS	16

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

Figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Copy Engine Logical Context Data [CHV]

Overall Context Layout

Context Layout

For [CHV], when Execlists are enabled, the Context Image for the copy engine consists of two 4K pages:

Register State context is explained in detail in “Register State Context” Section.

Register/State Context

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in Execution List mode of scheduling. Ring Buffer registers are directly programmed in Ring Buffer mode of scheduling.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information, etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

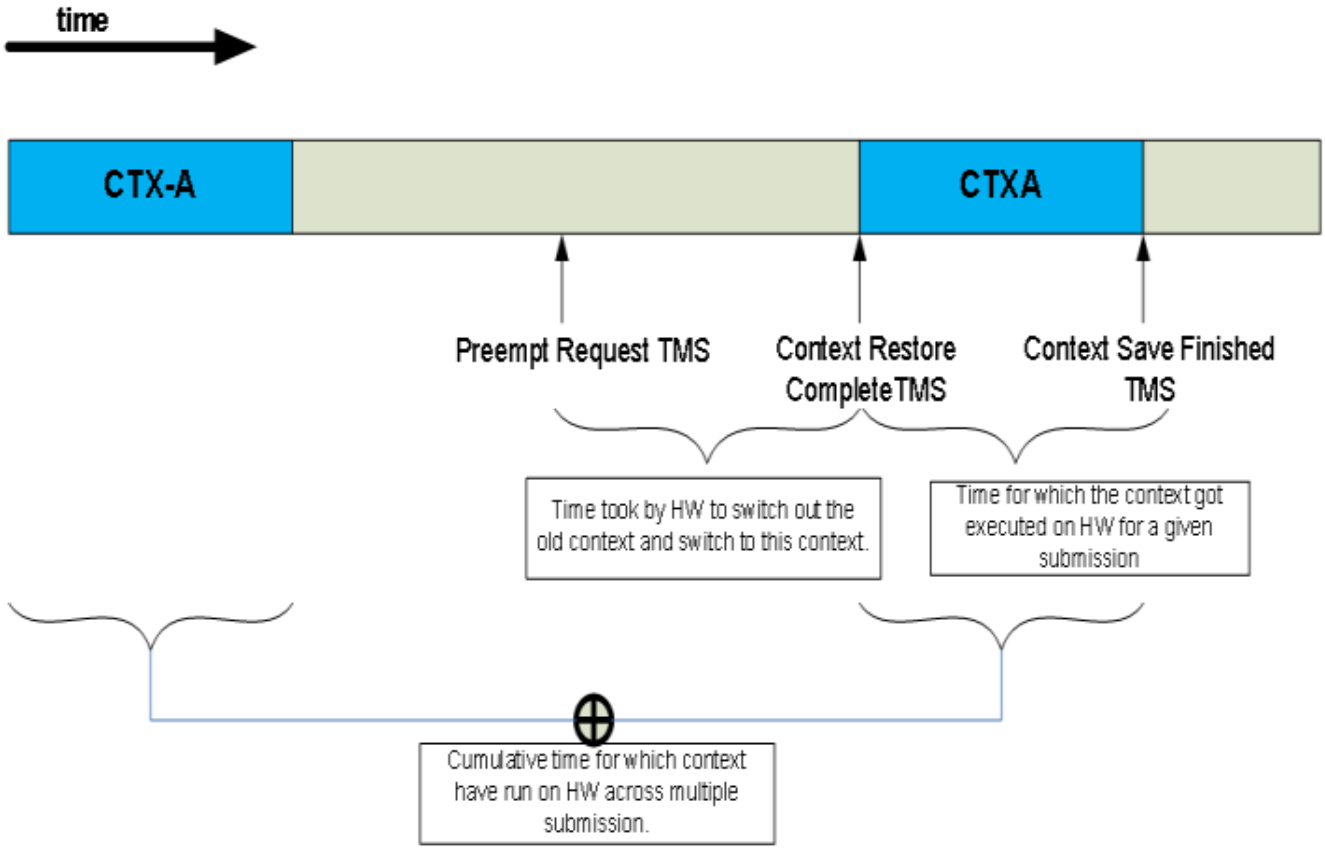
Description			# of DW
NOOP		BCS	1
Load_Register_Immediate header	0x1100_1015	BCS	1
Context Control	0x22244	BCS	2
Ring Head Pointer Register	0x22034	BCS	2
Ring Tail Pointer Register	0x22030	BCS	2
RING_BUFFER_START	0x22038	BCS	2
RING_BUFFER_CONTROL	0x2203C	BCS	2
Batch Buffer Current Head Register	0x22140	BCS	4
Batch Buffer State Register	0x22110	BCS	2
SECOND_BB_ADDR	0x22114	BCS	4
SECOND_BB_STATE	0x22118	BCS	2
NOOP		BCS	8
NOOP		BCS	1
Load_Register_Immediate header	0x1100_100F	BCS	1
CTX_TIMESTAMP	0x223A8	BCS	2
PDP3_UDW	0x 2228C	BCS	2
PDP3_LDW	0x 22288	BCS	2
PDP2_UDW	0x 22284	BCS	2
PDP2_LDW	0x 22280	BCS	2
PDP1_UDW	0x 2227C	BCS	2
PDP1_LDW	0x 22278	BCS	2
PDP0_UDW	0x 22274	BCS	2
PDP0_LDW	0x 22270	BCS	2
NOOP		BCS	12
NOOP		BCS	1
Load_Register_Immediate header	0x1100_1001	BCS	1
BCS_BTILEY	0x22200	BCS	2
NOOP		BCS	12

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

Figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Video Enhancement Logical Context Data

Overall Context Layout

Context Layout

For [CHV], when Execlists are enabled, the Context Image for the Video Enhancement engine consists of two 4K pages:

- Per-Process HW Status Page (4K)
- Register State Context

When Execlists are disabled (ring buffer mode of scheduling) there is no concept of context for workloads submitted to media engine and hence there is no context image.

Register State context is explained in detail in the “Register State Context” Section.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size.

Programming Note	
Context:	Submitting a context to HW for execution
The last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.	

Description			# of DW
NOOP		VECS	1
Load_Register_Immediate header	0x1100_1015	VECS	1
Context Control	0x1A244	VECS	2
Ring Head Pointer Register	0x1A034	VECS	2
Ring Tail Pointer Register	0x1A030	VECS	2
RING_BUFFER_START	0x1A038	VECS	2
RING_BUFFER_CONTROL	0x1A03C	VECS	2
Batch Buffer Current Head Register	0x1A140	VECS	4
Batch Buffer State Register	0x1A110	VECS	2
SECOND_BB_ADDR	0x1A114	VECS	4
SECOND_BB_STATE	0x1A118	VECS	2
NOOP		VECS	8
NOOP		VECS	1
Load_Register_Immediate header	0x1100_100F	VECS	1
CTX_TIMESTAMP	0x1A3A8	VECS	2
PDP3_UDW	0x 1A28C	VECS	2
PDP3_LDW	0x 1A288	VECS	2
PDP2_UDW	0x 1A284	VECS	2
PDP2_LDW	0x 1A280	VECS	2
PDP1_UDW	0x 1A27C	VECS	2
PDP1_LDW	0x 1A278	VECS	2
PDP0_UDW	0x 1A274	VECS	2
PDP0_LDW	0x 1A270	VECS	2
NOOP		VECS	12
NOOP		VECS	16

Memory Data Formats

This chapter describes the attributes associated with the memory-resident data objects operated on by the graphics pipeline. This includes object types, pixel formats, memory layouts, and rules/restrictions placed on the dimensions, physical memory location, pitch, alignment, etc. with respect to the specific operations performed on the objects.

Unsigned Normalized (UNORM)

An unsigned normalized value with *n* bits is interpreted as a value between 0.0 and 1.0. The minimum value (all 0's) is interpreted as 0.0, the maximum value (all 1's) is interpreted as 1.0. Values in between are equally spaced. For example, a 2-bit UNORM value would have the four values 0, 1/3, 2/3, and 1.

If the incoming value is interpreted as an n-bit integer, the interpreted value can be calculated by dividing the integer by 2^n-1.

Gamma Conversion (SRGB)

Gamma conversion is only supported on UNORM formats. If this flag is included in the surface format name, it indicates that a reverse gamma conversion is to be done after the source surface is read, and a forward gamma conversion is to be done before the destination surface is written.

Signed Normalized (SNORM)

Programming Note	
Context:	Signed normalized value in memory data formats.
A signed normalized value with *n* bits is interpreted as a value between -1 and +1.0. If the incoming value is interpreted as a 2's-complement n-bit integer, the interpreted value can be calculated by dividing the integer by 2^{n-1}-1. The most negative value of -2^{n-1} will result in a value slightly smaller than -1.0. This value is clamped to -1.0; thus, there are two representations of -1.0 in SNORM format.	

Unsigned Integer (UINT/USCALED)

The UINT and USCALED formats interpret the source as an unsigned integer value with *n* bits with a range of 0 to 2^n-1.

The UINT formats copy the source value to the destination (zero-extending if required), keeping the value as an integer.

The USCALED formats convert the integer into the corresponding floating point value (e.g., 0x03 --> 3.0f). For 32-bit sources, the value is rounded to nearest even.

Signed Integer (SINT/SSCALED)

A signed integer value with *n* bits is interpreted as a 2's complement integer with a range of -2^{n-1} to $+2^{n-1}$-1.

The SINT formats copy the source value to the destination (sign-extending if required), keeping the value as an integer.

The SSCALED formats convert the integer into the corresponding floating point value (e.g., 0xFFFD --> -3.0f). For 32-bit sources, the value is rounded to nearest even.

Floating Point (FLOAT)

Refer to IEEE Standard 754 for Binary Floating-Point Arithmetic. The IA-32 Intel (R) Architecture Software Developer's Manual also describes floating point data types .

64-bit Floating Point

Bit	Description
63	**Sign (s)**
62:52	**Exponent (e)** Biased Exponent
51:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == b'11..11' and f != 0, then v is NaN regardless of s
- if e == b'11..11' and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < b'11..11', then v = $(-1)^s*2^{(e-1023)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-1022)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s$*0 (signed zero)

32-bit Floating Point

Bit	Description
31	**Sign (s)**
30:23	**Exponent (e)** Biased Exponent
22:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == 255 and f != 0, then v is NaN regardless of s
- if e == 255 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 255, then v = $(-1)^s*2^{(e-127)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-126)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

16-bit Floating Point

Bit	Description
15	**Sign (s)**
14:10	**Exponent (e)** Biased Exponent
9:0	**Fraction (f)** Does not include "hidden one"

The value of this data type is derived as:

- if e == 31 and f != 0, then v is NaN regardless of s
- if e == 31 and f == 0, then v = $(-1)^s$*infinity (signed infinity)
- if 0 < e < 31, then v = $(-1)^s*2^{(e-15)}*(1.f)$
- if e == 0 and f != 0, then v = $(-1)^s*2^{(e-14)}*(0.f)$ (denormalized numbers)
- if e == 0 and f == 0, then v = $(-1)^s*0$ (signed zero)

The following table represents relationship between 32 bit and 16 bit floating point ranges:

flt32 exponent	Unbiased exponent	Normalization	flt16 exponent	flt16 fraction
255				
254	127			
...				
127+16	16	Infinity	31	1.1111111111
127+15	15	Max exponent	30	1.xxxxxxxxxx
127	0		15	1.xxxxxxxxxx
113	-14	Min exponent	1	1.xxxxxxxxxx
112		Denormalized	0	0.1xxxxxxxxx
111		Denormalized	0	0.01xxxxxxxx
110		Denormalized	0	0.001xxxxxxx
109		Denormalized	0	0.0001xxxxxx
108		Denormalized	0	0.00001xxxxx
107		Denormalized	0	0.000001xxxx
106		Denormalized	0	0.0000001xxx
115		Denormalized	0	0.00000001xx
114		Denormalized	0	0.000000001x
113		Denormalized	0	0.0000000001
112		Denormalized	0	0.0
...				
0			0	0.0

Conversion from the 32-bit floating point format to the 16-bit format should be done with round to nearest even.

11-bit Floating Point

Bits	Description
10:6	**Exponent (e):** Biased exponent (the bias depends on e)
5:0	**Fraction (f):** Fraction bits to the right of the binary point

The value v of an 11-bit floating-point number is calculated from e and f as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

There is no sign bit and negative values are not represented.

The 11-bit floating-point format has one more bit of fractional precision than the 10-bit floating-point format.

The maximum representable finite value is 1.111111b * 2^{15} = FE00h = 65024.

10-bit Floating Point

Bits	Description
9:5	**Exponent (e):** Biased exponent (the bias depends on e)
4:0	**Fraction (f):** Fraction bits to the right of the binary point

The value v of a 10-bit floating-point number is calculated from e and f as:

- if e == 31 and f != 0 then v = NaN
- if e == 31 and f == 0 then v = +infinity
- if 0 < e < 31, then v = $2^{(e-15)}$*(1.f)
- if e == 0 and f != 0, then v = $2^{(e-14)}$*(0.f) (denormalized numbers)
- if e == 0 and f == 0, then v = 0 (zero)

There is no sign bit and negative values are not represented.

The maximum representable finite value is 1.11111b * 2^{15} = FC00h = 64512.

Shared Exponent

The R9G9B9E5_SHAREDEXP format contains three channels that share an exponent. The three fractions assume an impled “0” rather than an implied “1” as in the other floating point formats. This format does not support infinity and NaN values. There are no sign bits, only positive numbers and zero can be represented. The value of each channel is determined as follows, where “f” is the fraction of the corresponding channel, and “e” is the shared exponent.

$v = (0.f)*2^{(e-15)}$

Bit	Description
31:27	**Exponent (e)** Biased Exponent
26:18	**Blue Fraction**
17:9	**Green Fraction**
8:0	**Red Fraction**

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 4: Configurations

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Configurations Overview 1

Configurations 1

Top Level Block Diagrams 1

Device Attributes 4

Steppings and Device IDs 7

Configurations Overview

The Intel "Gen" Graphics Architecture was first introduced to the market in 2004. Since that time, the architecture and its implementation have evolved to add many new features, increase performance, and improve power efficiency.

Configurations

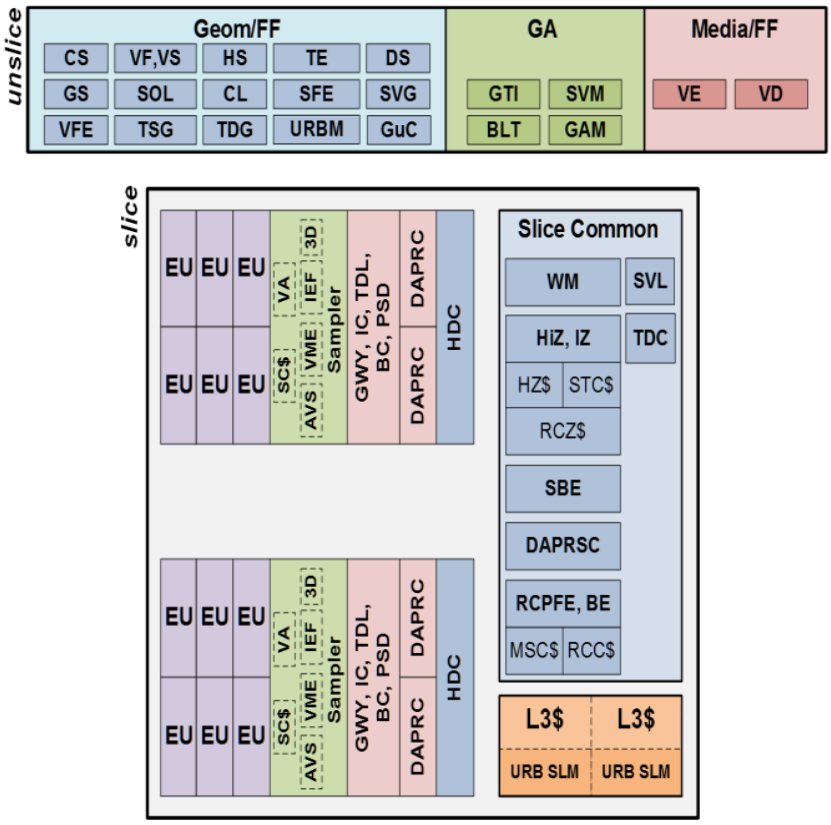
This chapter contains configurations details as described in the following sections:

- Top Level Block Diagrams
- Device Attributes
- Steppings & Device IDs

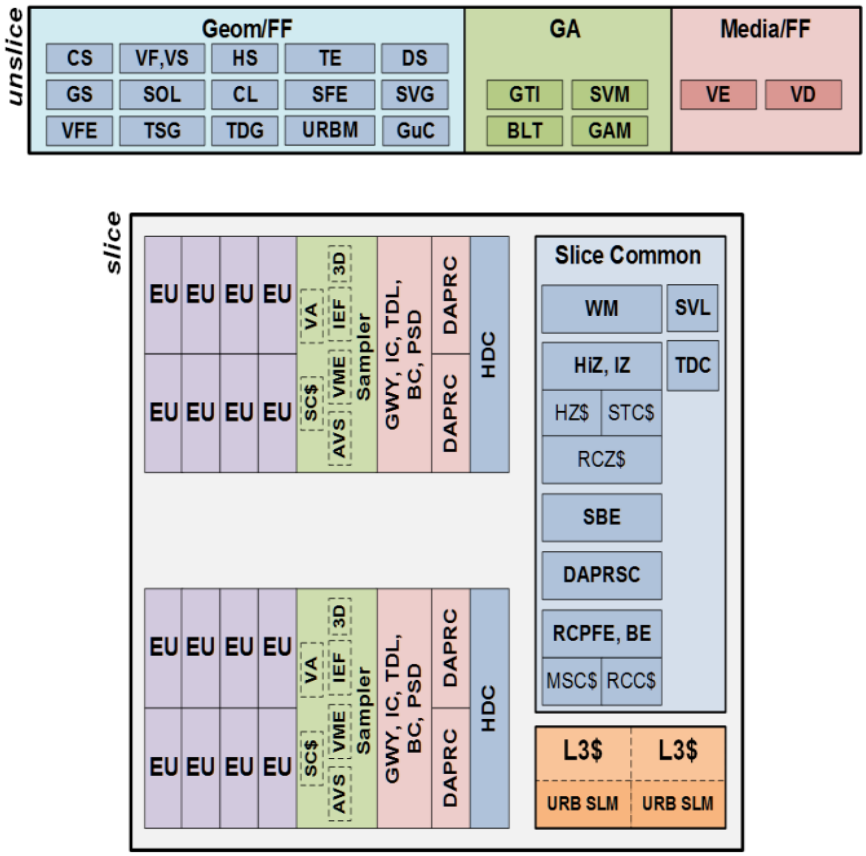
Top Level Block Diagrams

The diagram below shows basic feature blocks of the Cherryview (CHV) and Braswell (BSW) graphics architecture:

12 EU (2x6) Configuration:



16 EU (2x8) Configuration:



The above diagrams are based on the following functional partitions:

(a) Geometry Fixed Functions (Geom/FF)

(b) Media Fixed Functions (Media/FF)

(c) Global Assets and GT Interface (GA)

(d) One or more Subslices (two shown)

(e) A Slice Common block

(f) An L3 Cache (L3$) block

Note that the combination of (a), (b), and (c) is typically referred to as the “unslice”, while a combination of (d), (e), and (f) is referred to as a compute “slice”.

The functionality in each of these groupings is further broken down as follows:

- Unslice – Fixed function pipelines for 3D, GPGPU, and Media operations, and interface to the outside world.
  - The 3D Geometry / Fixed Function (Geom/FF) block consisting of:
    - 3D fixed function pipeline (CS, VFVS, HS, TE, DS, GS, SOL, SL, SFE, SVG)
    - Video Front-End unit (VFE)
    - Thread Spawner unit (TSG) and the global Thread Dispatcher unit (TDG)

        - Unified Return Buffer Manager (URBM)
    - Media fixed function assets:
        - Video Decode (VD) Box
        - Video Encode (VE) Box
        - Wireless Display (WD) BOX
    - The Global Assets (GA) block as the primary interface and memory stream gateway to the outside world, consisting of:
        - GT Interface (GTI)
        - State Variable Manager (SVM)
        - Blitter (BLT)
        - Graphics Arbiter (GAM)
- Subslice (three shown) – A compute unit with supporting fixed- or shared-function assets sufficient for the EU capability.
    - A bank of Execution Units (EUs) – eight per subslice shown
    - Sampler, supporting both media and 3D functions
    - Gateway (GWY)
    - Instruction cache (IC)
    - Local Thread Dispatcher (TDL)
    - Barycentric Calculator (BC)
    - Pixel Shader Dispatcher (PSD)
    - Data Cluster (HDC)
    - Dataport Render Cache (DAPRC) - two per subslice
- Slice Common – Scalable fixed function assets which support the compute horsepower provided two or more subslices.
    - 3D Fixed Function:
        - Windower/Mask unit (WM)
        - Plane-Z, Hi-Z (HZ) and Intermediate Z (IZ)
        - Setup Backend (SBE)
        - Pixel backend units
        - 3D stream caches for color, multi-sample surface, iz, and stencil (RCC$, MSC$, HZ$, RCZ$, STC$)
    - Media Fixed Function:
        - DAPRSC
        - RCPFE, BE
        - SVL
        - TDC
- L3 Cache – backing L3 cache for certain memory streams emanating from subslices.
    - L3 Data cache with support for data, URB, and shared local memory (SLM)

Device Attributes

Product Configuration Attribute Table		
Product Family	**CHV / BSW**	
Architectural Name	**2x6**	**2x8**
Slice count	1	1
Subslice Count	2	2
EU/Subslice	6	8
EU count (total)	12	16
Thread Count	7	7
Thread Count (Total)	84	112
FLOPs/Clk - Half Precision, MAD (peak)	384	512
FLOPs/Clk - Single Precision, MAD (peak)	192	256
FLOPs/Clk - Double Precision, MAD (peak)	24	32
Unslice clocking (coupled/decoupled from Cr slice)	coupled	coupled
GTI / Ring Interfaces	1	1
GTI bandwidth (bytes/unslice-clk)	64: R	64: R
	64: W	64: W
L3 Cache, total size (bytes)	384K	384K
L3 Cache, bank count	2	2
L3 Cache, bandwidth (bytes/clk)		
L3 Cache, D$ Size (Kbytes)	192K-256K	192K-256K
URB Size (kbytes)	64K-192K	64K-192K
SLM Size (kbytes)	0, 128K	0, 128K
LLC/L4 size (bytes)	N/A	N/A
Instruction Cache (IC, bytes)	2x 48K	2x 48K
Color Cache (RCC, bytes)	24K	24K
MSC Cache (MSC, bytes)	12K	12K
HiZ Cache (HZC, bytes)	8K	8K
Z Cache (RCZ, bytes)	16K	16K
Stencil Cache (STC, bytes)	4K	4K
L1 Texture Cache (bytes)	2x 32K	2x 32K
MT Texture Cache (bytes)	2x 8K	2x 8K
FMAD, SP (ops/EU/clk)	8	8
FMUL, SP (ops/EU/clk)	8	8
FADD, SP (ops/EU/clk)	8	8
MIN,MAX, SP (ops/EU/clk)	8	8
CMP, SP (ops/EU/clk)	8	8

Configurations

Product Configuration Attribute Table		
INV, SP (ops/EU/clk)	2	2
SQRT, SP (ops/EU/clk)	2	2
RSQRT, SP (ops/EU/clk)	2	2
LOG, SP (ops/EU/clk)	2	2
EXP, SP (ops/EU/clk)	2	2
POW, SP (ops/EU/clk)	1	1
IDIV, SP (ops/EU/clk)	1-6	1-6
TRIG, SP (ops/EU/clk)	2	2
FDIV, SP (ops/EU/clk)	1	1
Data Ports (HDC)	2	2
L3 Load/Store - same addresses within msg (dwords/clk)	2x 32	2x 32
L3 Load/Store - unique addresses within msg (dwords/clk)		
SLM Load//Store - same addresses within msg (dwords/clk)		
SLM Load//Store - unique addresses within msg (dwords/clk)		
Atomic, Local 32b - same addresses within msg (dwords/clk)	2x 16	2x 16
Atomic, Global 32b - unique addresses within msg (dwords/clk)	2x 16	2x 16
Geometry pipes	1	1
Samplers (3D)	2	2
Texel Rate, point, 32b (tex/clk)	8	8
Texel Rate, point, 64b (tex/clk)	8	8
Texel Rate, point, 128b (tex/clk)	8	8
Texel Rate, bilinear, 32b (tex/clk)	8	8
Texel Rate, bilinear, 64b (tex/clk)	8	8
Texel Rate, bilinear, 128b (tex/clk)	2	2
Texel Rate, trilinear, 32b (tex/clk)	4	4
Texel Rate, trilinear, 64b (tex/clk)	2	2
Texel Rate, trilinear, 128b (tex/clk)	1	1
Texel Rate, aniso 2x, 32b (tex/clk)	2	2
Texel Rate, aniso 4x, 32b (tex/clk)	1	1
Texel Rate, ansio 8x, 32b (tex/clk)	0.5	0.5
Texel Rate, ansio 16x, 32b (tex/clk)	0.25	0.25
HiZ Rate, (ppc)	32	32
IZ Rate, (ppc)	16	16
Stencil Rate (ppc)	32	32
(500 MHz, DDR-1600 or eDRAM; Range depends on dynamic compression ratio)		
Pixel Rate, fill, 32bpp (pix/clk, RCC hit)	4	4

Product Configuration Attribute Table		
Pixel Rate, fill, 32bpp (pix/clk, LLC hit @ 1.0x unslice clk)	N/A	N/A
Pixel Rate, fill, 32bpp (pix/clk, LLC hit, @ 1.5x unslice clk)	N/A	N/A
Pixel Rate, fill, 32bpp (pix/clk, memory, @ 1.0x unslice clk)	4	4
Pixel Rate, fill, 32bpp (pix/clk, memory, @ 1.5x unslice clk)	N/A	N/A
(500 MHz, DDR-1600 or eDRAM; Range depends on dynamic compression ratio)		
Pixel Rate, blend, 32bpp (p/clk, RCC hit)	4	4
Pixel Rate, blend, 32bpp (p/clk, LLC hit, @ 1.0x unslice clk)	N/A	N/A
Pixel Rate, blend, 32bpp (p/clk, LLC hit, @ 1.5x unslice clk)	N/A	N/A
Pixel Rate, blend, 32bpp (pix/clk, memory, @ 1.0x unslice clk)	4	4
Pixel Rate, blend, 32bpp (pix/clk, memory, @ 1.5x unslice clk)	N/A	N/A
Samplers (media)	2	2
VDBox Instances	1	1
VEBox Instances	1	1
SFC Instances	N/A	N/A
WGBox Instances	N/A	N/A

Programming Note	
Context:	Device Attributes
Unslice clock and slice clock are coupled; independent frequency control is not supported.	

Steppings and Device IDs

GT SKUs and Device IDs: The following table lists current GT Die / Packages for Cherry Trail (CHT) and Braswell (BSW).

Program Name	Gen	GT SKU	Total EUs	CPU / GT Stepping	GT Device2 Native ID	GT Device2 Revision ID	Comments / Description
CHT	Gen8LP	2x6	12	C0	0x22B0	0x20, 0x22, or 0x23 (depending on packaging)	Production Devices: Atom x5 series
CHT	Gen8LP	2x8	16	C0	0x22B0	0x20, 0x22, or 0x23 (depending on packaging)	Production Devices: Atom x7 series
BSW	Gen8LP	2x6	12	C0	0x22B1	0x21	Production Devices: Celeron QC and and Celeron DC series
BSW	Gen8LP	2x8	16	C0	0x22B1	0x21	Production Devices: Pentium QC series

Programming Note	
Project:	CHV
Context:	Steppings and Device IDs
See the DID [CHV] register.	

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 5: Memory Views

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Introduction **1**
Memory Views Glossary 1
GPU Memory Interface **1**
Global Arbitration **2**
Graphics Memory Interface Functions **2**
Graphics Memory Clients 3
Graphics Memory Addressing Overview 4
Graphics Address Path 4
Graphics Memory Paths 5
Graphics Memory Address Spaces 6
Address Tiling Function Introduction **6**
Linear vs Tiled Storage 7
Tile Formats 10
Tile-X Legacy Format 10
Tile-Y Legacy Format 11
Tiling Algorithm 12
Tiled Channel Select Decision 14
Tiling Support 14
Tiled (Fenced) Regions 15
Tiled Surface Parameters 16
Tiled Surface Restrictions 17
Per-Stream Tile Format Support 20
Main Memory 21
Optimizing Main Memory Allocation 21
Application of the Theory (Page Coloring) 22
3D Color and Depth Buffers 22
Media/Video 22
Physical Graphics Address Types 22
Graphics Translation Tables **23**
Virtual Memory 24
GFX Page Tables 24

Page Table Modes 24
Per Process GTT 25
PPGTT for 32b Virtual Address 25
Walk with 64KB Page 26
Walk with 2MB Page 27
Walk with 1GB Page 28
PPGTT for Standard Context (64b VA) 29
Walk with 64KB Page 30
Walk with 2MB Page 30
Walk with 1GB Page 31
Global GTT 32
Page Table Entry 32
Page Walk 33
GTT Cache 34
GFX Page Walker (GAM) 34
Context Definition for GFX Page Walker 34
Context Definition Delivery 36
Element Descriptor Register 37
PDP0/PML4/PASID Descriptor Register 39
PDP1 Descriptor Register 39
PDP2 Descriptor Register 40
PDP3 Descriptor Register 40
List of Registers and Command Streamers 41
Updating Page Table Pointers (aka PD Load) 43
Page Walker (GAM) Reset 44
Legacy Context 44
Full Walk 45
TLB Caching and Management 46
TLB Caches 47
Context Cache - CC 47
PASID Cache - PC 47
Intermediate Page Walk Caches (PML4, PDP, PD) – PWC 48
TLB – Final Page Entry 48

TLB Entry Content 49
TLB Accessed and Dirty Flags 51
Updating A/D Bits 51
PAT (IA32e) 56
PAT in Context Table Entry 56
Memory Types and Applicability to GFX 56
MTRR Ranges 59
Memory Type Selection and Priority 59
Design-Specific Memory Types 59
Memory Object Control State (Surface) 60
Architectural Memory Types 61
Page Walker Access and Memory Types 64
Page Walker Memory Types 64
Error Cases 65
Replacement 65
Invalidations of TLB 65
Optional Invalidations 66
Faulting 67
Page Faults 67
Memory Types and Cache Interface 68
Memory Object Control State (MOCS) 68
MOCS Registers 69
Page Walker Access and Memory Types 69
Page Walker Memory Types 69
Gen8 Memory Typing for Paging 70
Error Cases 71
Common Surface Formats 72
Non-Video Surface Formats 72
Surface Format Naming 72
Intensity Formats 72
Luminance Formats 72
R1_UNORM (same as R1_UINT) and MONO8 73
Palette Formats 74

P4A4_UNORM 74
A4P4_UNORM 74
P8A8_UNORM 75
A8P8_UNORM 75
P8_UNORM 76
P2_UNORM 76
Compressed Surface Formats 77
ETC1_RGB8 77
ETC2_RGB8 and ETC2_SRGB8 80
T mode 81
H mode 83
Planar mode 85
EAC_R11 and EAC_SIGNED_R11 87
ETC2_RGB8_PTA and ETC2_SRGB8_PTA 89
Differential Mode 89
T and H Modes 89
Planar Mode 89
ETC2_EAC_RGBA8 and ETC2_EAC_SRGB8_A8 90
EAC_RG11 and EAC_SIGNED_RG11 91
FXT Texture Formats 93
Overview of FXT1 Formats 93
FXT1 CC_HI Format 93
CC_HI Block Encoding 94
CC_HI Block Decoding 94
FXT1 CC_CHROMA Format 95
CC_CHROMA Block Encoding 96
CC_CHROMA Block Decoding 96
FXT1 CC_MIXED Format 98
CC_MIXED Block Encoding 98
CC_MIXED Block Decoding 98
FXT1 CC_ALPHA Format 102
CC_ALPHA Block Encoding 102
CC_ALPHA Block Decoding 103

DXT/BC1-3 Texture Formats 106
Opaque and One-bit Alpha Textures (DXT1/BC1) 107
Opaque Textures (DXT1_RGB) 109
Compressed Textures with Alpha Channels (DXT2-5 / BC2-3) 109
BC4 112
BC5 113
BC6H 116
Field Definition 116
Endpoint Computation 132
Palette Color Computation 132
Texel Selection 133
ONE Mode 133
TWO Mode 133
BC7 135
Field Definition 135
Endpoint Computation 143
Palette Color Computation 144
Texel Selection 144
ONE Mode 144
TWO Mode 145
THREE Mode 147
Adaptive Scalable Texture Compression (ASTC) 149
ASTC Fundamentals 149
Background 149
New Surface Formats for ASTC Texture 151
ASTC File Format and Memory Layout 154
ASTC Header Data Structure and Amendment 154
Data Layout in ASTC Compression File 155
Total ASTC Data Block Layout in All Mipmap Levels 156
Data Layout in Memory for All Mipmap Levels 156
ASTC Data Structure 159
Layout and Description of Block Data 160
Partitioning 160

Index Mode 161
Index Planes 165
Index Infill Procedure 166
Color Endpoint Mode 166
Color Endpoint Data Size Determination 168
Void-Extent Blocks 169
Decoding Process 170
Overview Decoding Flow 170
Integer Sequence Encoding 173
Endpoint Unquantization 174
LDR Endpoint Decoding 175
HDR Endpoint Decoding 178
HDR Endpoint Mode 2 (HDR Luminance, Large Range) 178
HDR Endpoint Mode 3 (HDR Luminance, Small Range) 179
HDR Endpoint Mode 7 (HDR RGB, Base+Scale) 180
HDR Endpoint Mode 11 (HDR RGB, Direct) 182
HDR Endpoint Mode 14 (HDR RGB, Direct + LDR Alpha) 185
HDR Endpoint Mode 15 (HDR RGB, Direct + HDR Alpha) 186
Restrictions on Number of Partitions Per Block 187
Index Decoding 187
Index Unquantization 187
Infill Process 189
Index Application 191
Dual-Plane Decoding 191
Partition Pattern Generation 192
Data Size Determination 194
3D Void-Extent Blocks 195
Illegal Encodings 195
Profile Support 196
Video Pixel/Texel Formats 197
Packed Memory Organization 197
Planar Memory Organization 197
Raw Format 200

Surface Memory Organizations 200
Display, Overlay, Cursor Surfaces 200
2D Render Surfaces 200
2D Monochrome Source 200
2D Color Pattern 201
3D Color Buffer (Destination) Surfaces 201
3D Depth Buffer Surfaces 202
3D Separate Stencil Buffer Surfaces 202
Surface Layout 203
Buffers 203
Structured Buffers 203
1D Surfaces 204
2D Surfaces 204
Computing MIP Level Sizes 205
Base Address for LOD Calculation 205
Minimum Pitch for MIPLAYOUT_RIGHT and Other Maps 207
Cartesian to Linear Address Conversion 207
Compressed Mipmap Layout 207
Surface Arrays 208
For All Surfaces 208
Multisampled Surfaces 208
Compressed Multisampled Surfaces 209
Physical MSS Surface 210
Uncompressed Multisampled Surfaces 210
Cube Surfaces 210
DirectX API Definition 210
Hardware Cube Map Layout 211
Restrictions 211
Cube Arrays 212
3D Surfaces 213
Minimum Pitch 215
Surface Padding Requirements 216
Alignment Unit Size 216

Sampling Engine Surfaces 217
Render Target and Media Surfaces 218

Introduction

The hardware supports three engines:

- The Render command streamer interfaces to 3D/IE and display streams.
- The Media command streamer interfaces to the fixed function media.
- The Blitter command streamer interfaces to the blit commands.

Software interfaces of all three engines are very similar and should only differ on engine-specific functionality.

Memory Views Glossary

Term	Definition
CHV, BSW	CherryView CPU/GFX platform. 8th generation processor graphics (Gen8).
IOMMU	I/O Memory Mapping unit
SVM	Shared Virtual Memory, implies the same virtual memory view between the IA cores and processor graphics.
Page Walker (GAM)	GFX page walker which handles page level translations between GFX virtual memory to physical memory domain.

GPU Memory Interface

GPU memory interface functions are divided into 4 different major sections:

- Global Arbitration
- Memory Interface Functions
- Page Translations (GFX Page Walker)
- Ring Interface Functions (GTI)

GT Interface functions are covered at a different chapter/HAS and not part of this documentation. The following documentation is meant for GFX arbitration paths in accessing to memory/cache interfaces and page translations and page walker functions.

Global Arbitration

The global memory arbitration fabric is meant to be a hierarchal memory fabric where memory accesses from different stages of the pipeline are consolidated to a single interface towards GT's connection to CPU's ring interface.

The arbitration on the fabric is programmable via a simple per pipeline stage priority levels.

Programming Note	
Context:	Global Memory Arbitration
Arbitration allows 4 levels of arbitration where each pipeline level can be put into these 4 levels. Each consolidation stage simply follows the 4-level arbitration with grace periods to allow ahead of the pipeline to get a higher share of the memory bandwidth.	

The final arbitration takes places in GAM between parallel compute engines. Each engine (in some cases major pipeline stages are also separated, i.e. Z vs Color vs L3 vs Fixed Functions) gets a count in a grace period where its accesses are counted against a global pool. If a particular engine (or pipeline stage) exhausts its max allowed, it is dropped to a lower priority and goes to fixed pipeline based prioritization. Once all counts are expired, the grace period completes and resets.

The count values are programmable via MMIO (i.e. *_MAX_REQ_COUNT) registers with defaults favoring the pipeline order.

Graphics Memory Interface Functions

The major role of an integrated graphics device's Memory Interface (MI) function is to provide various client functions access to "graphics" memory used to store commands, surfaces, and other information used by the graphics device. This chapter describes the basic mechanisms and paths by which graphics memory is accessed.

Information not presented in this chapter includes:

- Microarchitectural and implementation-dependent features (e.g., internal buffering, caching, and arbitration policies).
- MI functions and paths specific to the operation of external (discrete) devices attached via external connections.
- MI functions essentially unrelated to the operation of the internal graphics devices, .e.g., traditional "chipset functions"
- GFX Page Walker and GT interface functions are covered in different chapters.

Graphics Memory Clients

The MI function provides memory access functionality to a number of external and internal graphics memory *clients*, as described in the table below.

Graphics Memory Clients

MI Client	Access Modes
Host Processor	Read/Write of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses
External PEG Graphics Device	Write-Only of Graphics Operands located in Main Memory via the Graphics Aperture. (This client is not described in this chapter).
Peer PCI Device	Write-Only of Graphics Operands located in Main Memory. Graphics Memory is accessed using Device 2 Graphics Memory Range Addresses (i.e., mapped by GTT). Note that DMI access to Graphics registers is not supported.
Coherent Read/Write (internal)	Internally-generated snooped reads/writes.
Command Stream (internal)	DMA Read of graphics commands and related graphics data.
Vertex Stream (internal)	DMA Read of indexed vertex data from Vertex Buffers by the 3D Vertex Fetch (VF) Fixed Function.
Instruction/State Cache (internal)	Read of pipelined 3D rendering state used by the 3D/Media Functions and instructions executed by the EUs.
Render Cache (internal)	Read/Write of graphics data operated upon by the graphics rendering engines (Blt, 3D, MPEG, etc.) Read of render surface state.
Sampler Cache (internal)	Read of texture (and other sampled surface) data stored in graphics memory.
Display/Overlay Engines (internal)	Read of display, overlay, cursor and VGA data.
Media Engines	Read and write of media content and media processing.
uController	Read/Write (DMA) functions for u-controller and scheduler.

Graphics Memory Addressing Overview

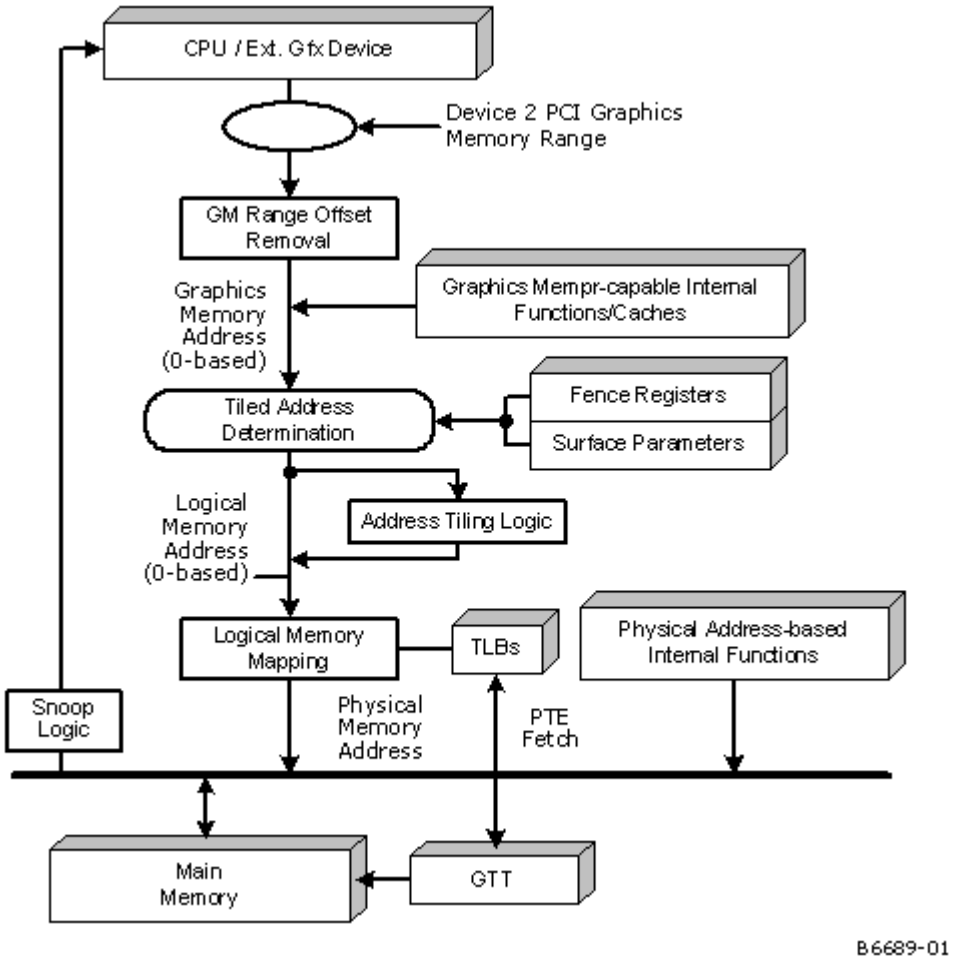
The Memory Interface function provides access to graphics memory (GM) clients. It accepts memory addresses of various types, performs a number of optional operations along *address paths*, and eventually performs reads and writes of graphics memory data using the resultant addresses. The remainder of this subsection will provide an overview of the graphics memory clients and address operations.

Graphics Address Path

Graphics Address Path shows the internal graphics memory address path, connection points, and optional operations performed on addresses. Externally-supplied addresses are normalized to zero-based *Graphics Memory* (*GM) addresses* (GM_Address). If the GM address is determined to be a tiled address (based on inclusion in a fenced region or via explicit surface parameters), *address tiling* is performed. At this point the address is considered a *Logical Memory address*, and is translated into a *Physical Memory address* via the GTT and associated TLBs. The physical memory location is then accessed.

CPU accesses to graphics memory are sent back on the ring to snoop. Hence pages that are mapped cacheable in the GTT will be coherent with the CPU cache if accessed through graphics memory aperture.

Graphics Memory Paths



The remainder of this chapter describes the basic features of the graphics memory address pipeline, namely Address Tiling, Logical Address Mapping, and Physical Memory types and allocation considerations.

Graphics Memory Address Spaces

The *Graphics Memory Address Spaces* table lists the five supported Graphics Memory Address Spaces. Note that the Graphics Memory Range Removal function is automatically performed to transform system addresses to internal, zero-based Graphics Addresses.

Due to a workaround, first 4KB of DSM has to be reserved for GFX hardware use during render engine execution.

Address Tiling Function Introduction

When dealing with memory operands (e.g., graphics surfaces) that are inherently rectangular in nature, certain functions within the graphics device support the storage/access of the operands using alternative (tiled) memory formats to increase performance. This section describes these memory storage formats, why and when they should be used, and the behavioral mechanisms within the device to support them.

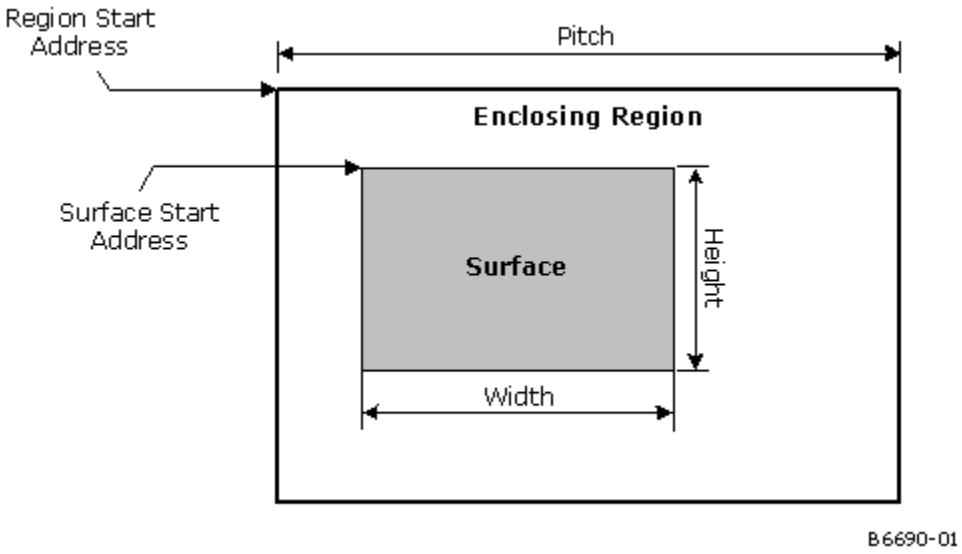
Legacy Tiling Modes:

- **TileY**: Used for most tiled surfaces when **TR_MODE**=TR_NONE.
- **TileX** : Used primarily for display surfaces.
- **TileW:** Used for Stencil surfaces.

Linear vs Tiled Storage

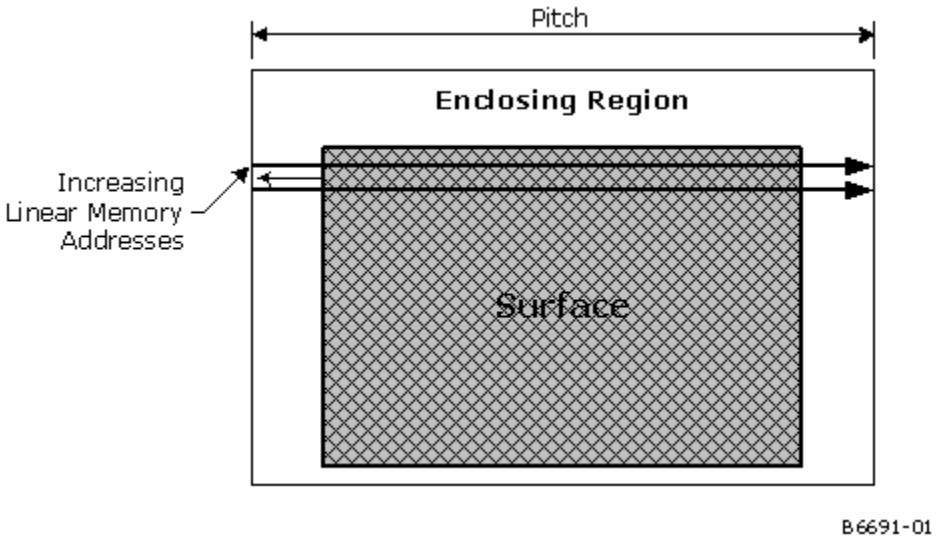
Regardless of the memory storage format, "rectangular" memory operands have a specific *width* and *height*, and are considered as residing within an enclosing rectangular region whose width is considered the *pitch* of the region and surfaces contained within. Surfaces stored within an enclosing region must have widths less than or equal to the region pitch (indeed the enclosing region may coincide exactly with the surface). *Rectangular Memory Operand Parameters* shows these parameters.

Rectangular Memory Operand Parameters



The simplest storage format is the *linear* format (see *Linear Surface Layout*), where each row of the operand is stored in sequentially increasing memory locations. If the surface width is less than the enclosing region's pitch, there will be additional memory storage between rows to accommodate the region's pitch. The pitch of the enclosing region determines the distance (in the memory address space) between vertically-adjacent operand elements (e.g., pixels, texels).

Linear Surface Layout

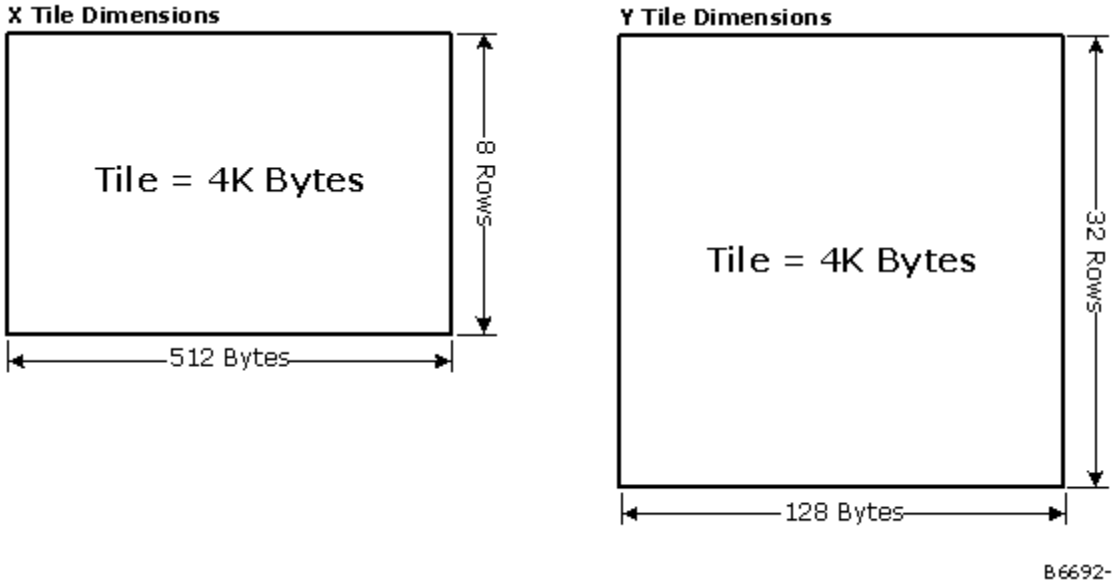


The linear format is best suited for 1-dimensional row-sequential access patterns (e.g., a display surface where each scanline is read sequentially). Here the fact that one object element may reside in a different memory page than its vertically-adjacent neighbors is not significant; all that matters is that horizontally-adjacent elements are stored contiguously. However, when a device function needs to access a 2D subregion within an operand (e.g., a read or write of a 4x4 pixel span by the 3D renderer, a read of a 2x2 texel block for bilinear filtering), having vertically-adjacent elements fall within different memory pages is to be avoided, as the page crossings required to complete the access typically incur increased memory latencies (and therefore lower performance).

One solution to this problem is to divide the enclosing region into an array of smaller rectangular regions, called memory *tiles*. Surface elements falling within a given tile will all be stored in the same physical memory page, thus eliminating page-crossing penalties for 2D subregion accesses within a tile and thereby increasing performance.

Tiles have a fixed 4KB size and are aligned to physical DRAM page boundaries. They are either 8 rows high by 512 bytes wide or 32 rows high by 128 bytes wide (see *Memory Tile Dimensions*). Note that the dimensions of tiles are irrespective of the data contained within – e.g., a tile can hold twice as many 16-bit pixels (256 pixels/row x 8 rows = 2K pixels) than 32-bit pixels (128 pixels/row x 8 rows = 1K pixels).

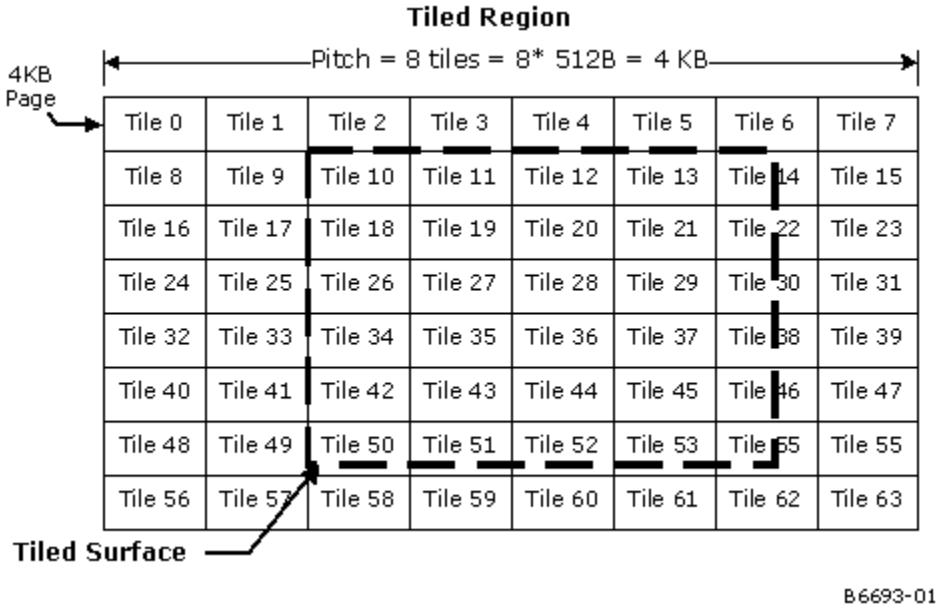
Memory Tile Dimensions



The pitch of a tiled enclosing region must be an integral number of tile widths. The 4KB tiles within a tiled region are stored sequentially in memory in row-major order.

The *Tiled Surface Layout* figure shows an example of a tiled surface located within a tiled region with a pitch of 8 tile widths (512 bytes * 8 = 4KB). Note that it is the *enclosing region* that is divided into tiles – the surface is not necessarily aligned or dimensioned to tile boundaries.

Tiled Surface Layout



Tile Formats

Multiple tile formats are supported by the Gen Core. The following sections define and describe these formats.

Tiling formats are controlled by programming the fields Tile_Mode and Tiled_Resource_Mode in the RENDER_SURFACE_STATE.

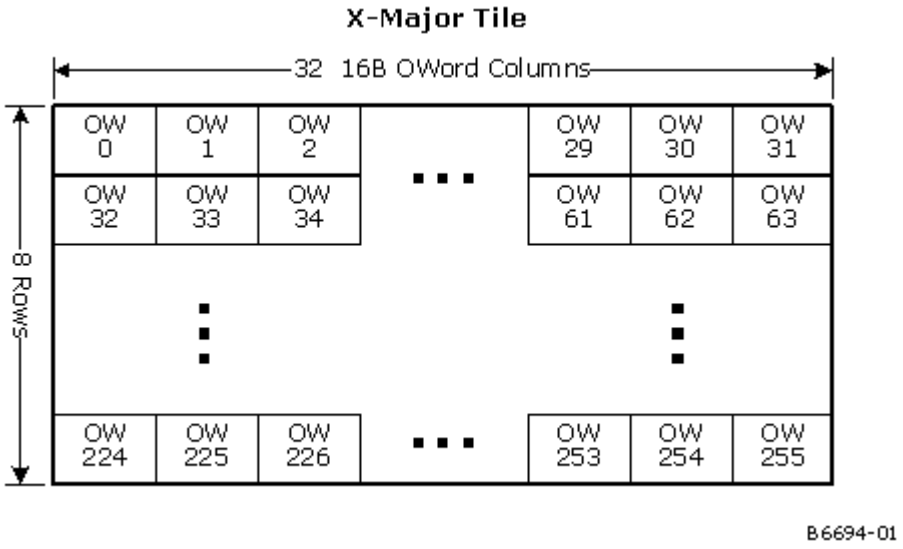
Tile-X Legacy Format

The legacy format Tile-X is a *X-Major* (row-major) storage of tile data units, as shown in the following figure. It is a 4KB tile which is subdivided into an 8-high by 32-wide array of 16-byte OWords . The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles. Note that an X-major tiled region with a tile pitch of 1 tile is actually stored in a linear fashion.

Tile-X format is selected for a surface by programming the Tiled_Mode field in RENDER_SURFACE_STATE to XMAJOR.

For 3D sampling operation, a surface using Tile-X layout is generally lower performance the organization of texels in memory.

Tile X-Tile (X-Major) Layout



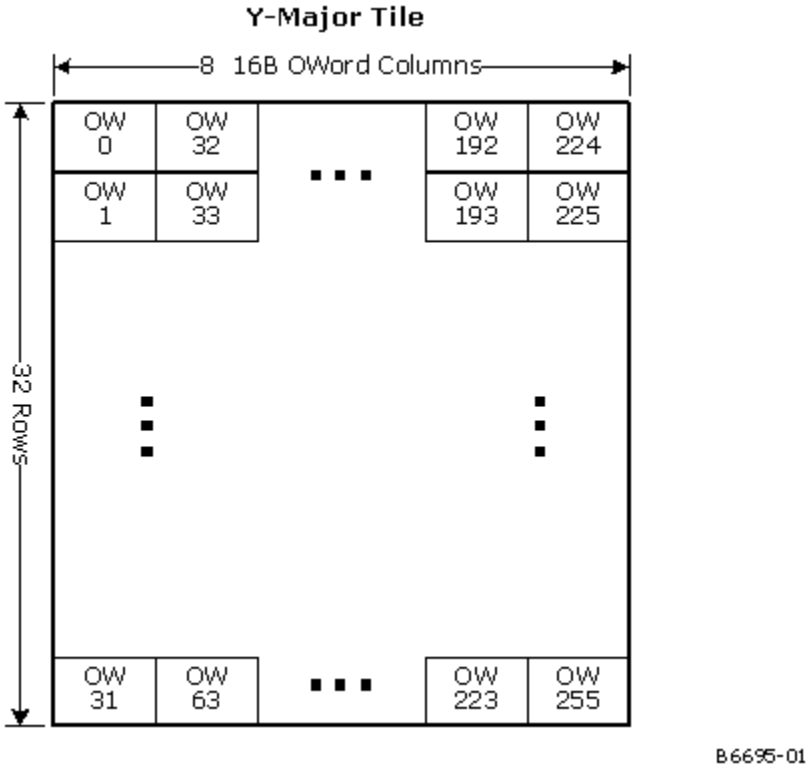
Tile-Y Legacy Format

The device supports Tile-Y legacy format which is *Y-Major* (column major) storage of tile data units, as shown in the following figure. A 4KB tile is subdivided 32-high by 8-wide array of OWords. The selection of tile direction only impacts the internal organization of tile data, and does not affect how surfaces map onto tiles.

Tile-Y surface format is selected by programming the Tile_Mode field in RENDER_SURFACE_STATE to YMAJOR.

Note that 3D sampling of a surface in Tile-Y format is usually has higher performance due to the layout of pixels.

Y-Major Tile Layout



Tiling Algorithm

The following pseudo-code describes the algorithm for translating a tiled memory surface in graphics memory to an address in logical space.

```
 Inputs:
     LinearAddress (offset into regular or LT aperture in terms of bytes)
     Pitch (in terms of tiles)
     WalkY (1 for Y and 0 for X)
     WalkW (1 for W and 0 for the rest)
 Static Parameters:
     TileH (Height of tile, 8 for X, 32 for Y, and 64 for W),
     TileW (Width of Tile in bytes, 512 for X, 128 for Y, and 64 for W)

 TileSize = TileH * TileW;
 RowSize = Pitch * TileSize;
 If ( Fenced ) {
     LinearAddress = LinearAddress - FenceBaseAddress;
     LinearAddrInTileW = LinearAddress div TileW;
     Xoffset_inTile = LinearAddress mod TileW;
     Y = LinearAddrInTileW div Pitch;
     X = LinearAddrInTileW mod Pitch + Xoffset_inTile;
 }

 // Internal graphics clients that access tiled memory already have the X, Y coordinates and
can start here.
 YOff_Within_Tile = Y mod TileH;
 XOff_Within_Tile = X mod TileW;
 TileNumber_InY = Y div TileH;
 TileNumber_InX = X div TileW;
 TiledOffsetY = RowSize * TileNumber_InY + TileSize * TileNumber_InX +
         TileH * 16 * (XOff_Within_Tile div 16) + YOff_Within_Tile * 16 + (XOff_Within_Tile
mod 16);
 TiledOffsetW = RowSize * TileNumber_InY + TileSize * TileNumber_InX +
         TileH * 8 * (XOff_Within_Tile div 8) +
         64 * (YOff_Within_Tile div 8) +
         32 * ((YOff_Within_Tile div 4) mod 2) +
         16 * ((XOff_Within_Tile div 4) mod 2) +
          8 * ((YOff_Within_Tile div 2) mod 2) +
          4 * ((XOff_Within_Tile div 2) mod 2) +
          2 * (YOff_Within_Tile mod 2) +
              (XOff_Within_Tile mod 2);
 TiledOffsetX = RowSize * TileNumber_InY + TileSize * TileNumber_InX + TileW *
YOff_Within_Tile + XOff_Within_Tile;
 TiledOffset = WalkW ? TiledOffsetW : (WalkY ? TiledOffsetY : TiledOffsetX);
 TiledAddress = Tiled ? (BaseAddress + TiledOffset) : (BaseAddress + Y*LinearPitch + X);
 TiledAddress = (Tiled &&
         (Address Swizzling for Tiled-Surfaces == 01)) ?
         (WalkW || WalkY) ?
         (TiledAddress div 128) * 128 +
         (((TiledAddress div 64) mod 2) ^
         ((TiledAddress div 512) mod 2)) +
         (TiledAddress mod 32)
         :
         (TiledAddress div 128) * 128 +
         (((TiledAddress div 64) mod 2) ^
         ((TiledAddress div 512) mod 2)
         ((TiledAddress Div 1024) mod2) +
         (TiledAddress mod 32)
         :
         TiledAddress;
```

Address Swizzling for Tiled-Surfaces is no longer used because the main memory controller has a more effective address swizzling algorithm.

For Address Swizzling for Tiled-Surfaces see ARB_MODE – Arbiter Mode Control register, ARB_CTL—Display Arbitration Control 1, and TILECTL - Tile Control register.

The Y-Major tile formats have the characteristic that a surface element in an even row is located in the same aligned 64-byte cacheline as the surface element immediately below it (in the odd row). This spatial locality can be exploited to increase performance when reading 2x2 texel squares for bilinear texture filtering, or reading and writing aligned 4x4 pixel spans from the 3D Render pipeline.

On the other hand, the X-Major tile format has the characteristic that horizontally-adjacent elements are stored in sequential memory addresses. This spatial locality is advantageous when the surface is scanned in row-major order for operations like display refresh. For this reason, the Display and Overlay memory streams only support linear or X-Major tiled surfaces. (Y-Major tiling is not supported by these functions.) This has the side effect that 2D- or 3D-rendered surfaces must be stored in linear or X-Major tiled formats if they are to be displayed. Non-displayed surfaces, e.g., “rendered textures”, can also be stored in Y-Major order.

Tiled Channel Select Decision

Before Gen8, there was a historical configuration control field to swizzle address bit[6] for in X/Y tiling modes. This was set in three different places: TILECTL[1:0], ARB_MODE[5:4], and DISP_ARB_CTL[14:13].

For Gen8 and subsequent generations, the swizzle fields are all reserved, and the CPU's memory controller performs all address swizzling modifications.

Tiling Support

The rearrangement of the surface elements in memory must be accounted for in device functions operating upon tiled surfaces. (Note that not all device functions that access memory support tiled formats). This requires either the modification of an element's linear memory address or an alternate formula to convert an element's X,Y coordinates into a tiled memory address.

However, before tiled-address generation can take place, some mechanism must be used to determine whether the surface elements accessed fall in a linear or tiled region of memory, and if tiled, what the tile region pitch is, and whether the tiled region uses X-Major or Y-Major format. There are two mechanisms by which this detection takes place: (a) an implicit method by detecting that the pre-tiled (linear) address falls within a "fenced" tiled region, or (b) by an explicit specification of tiling parameters for surface operands (i.e., parameters included in surface-defining instructions).

The following table identifies the tiling-detection mechanisms that are supported by the various memory streams.

Access Path	Tiling-Detection Mechanisms Supported
Processor access through the Graphics Memory Aperture	Fenced Regions
3D Render (Color/Depth Buffer access)	Explicit Surface Parameters
Sampled Surfaces	Explicit Surface Parameters
Blt operands	Explicit Surface Parameters
Display and Overlay Surfaces	Explicit Surface Parameters

Tiled (Fenced) Regions

The only mechanism to support the access of surfaces in tiled format by the host or external graphics client is to place them within “fenced” tiled regions within Graphics Memory. A fenced region is a block of Graphics Memory specified using one of the sixteen FENCE device registers. (See *Memory Interface Registers* for details). Surfaces contained within a fenced region are considered tiled from an external access point of view. Note that fences cannot be used to untile surfaces in the PGM_Address space since external devices cannot access PGM_Address space. Even if these surfaces (or any surfaces accessed by an internal graphics client) fall within a region covered by an enabled fence register, that enable will be effectively masked during the internal graphics client access. Only the explicit surface parameters described in the next section can be used to tile surfaces being accessed by the internal graphics clients.

Restriction: Each FENCE register (if its Fence Valid bit is set) defines a Graphics Memory region ranging from 4KB to the aperture size. The region is considered rectangular, with a pitch in tile widths from 1 tile width (128B or 512B) to 512 tile X widths (512 * 512B = 256KB) and 2048 tile Y widths (2048 * 128B = 256KB). Note that fenced regions must not overlap, or operation is UNDEFINED.

Context: Tiled (Fenced) Regions

Restriction: Also included in the FENCE register is a Tile Walk field that specifies which tile format applies to the fenced region.

Context: Tiled (Fenced) Regions

Tiled Surface Parameters

Internal device functions require explicit specification of surface tiling parameters via information passed in commands and state. This capability is provided to limit the reliance on the fixed number of fence regions.

The following table lists the surface tiling parameters that can be specified for 3D Render surfaces (Color Buffer, Depth Buffer, Textures, etc.) via SURFACE_STATE.

Surface Parameter	Description
Tiled Surface	If ENABLED, the surface is stored in a tiled format. If DISABLED, the surface is stored in a linear format.
Tile Walk	If Tiled Surface is ENABLED, this parameter specifies whether the tiled surface is stored in Y-Major or X-Major tile format.
Base Address	Additional restrictions apply to the base address of a Tiled Surface vs. that of a linear surface.
Pitch	Pitch of the surface. Note that, if the surface is tiled, this pitch must be a multiple of the tile width.

Tiled Surface Restrictions

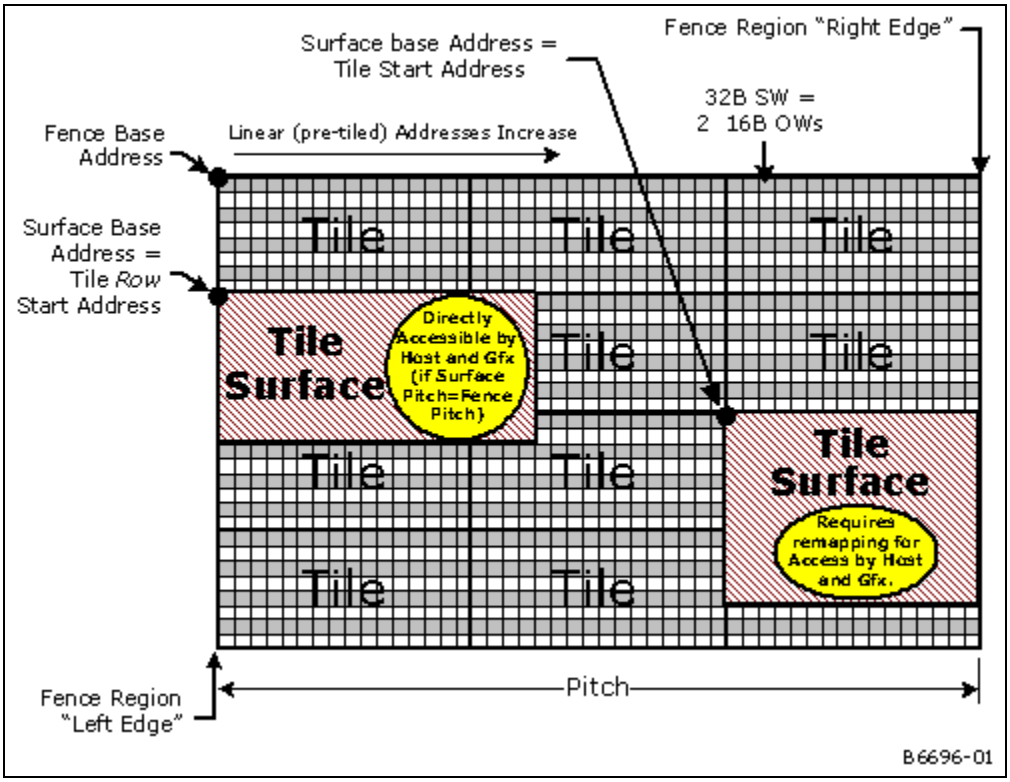
Additional restrictions apply to the Base Address and Pitch of a surface that is tiled. In addition, restrictions for tiling via SURFACE_STATE are subtly different from those for tiling via fence regions. The most restricted surfaces are those that will be accessed both by the host (via fence) and by internal device functions. An example of such a surface is a tiled texture that is initialized by the CPU and then sampled by the device.

The tiling algorithm for internal device functions is different from that of fence regions. Internal device functions always specify tiling in terms of a surface. The surface must have a base address, and this base address is not subject to the tiling algorithm. Only *offsets* from the base address (as calculated by X, Y addressing within the surface) are transformed through tiling. The base address of the surface must therefore be 4KB-aligned. This forces the 4KB tiles of the tiling algorithm to exactly align with 4KB device pages once the tiling algorithm has been applied to the offset. The width of a surface must be less than or equal to the surface pitch. There are additional considerations for surfaces that are also accessed by the host (via a fence region).

Fence regions have no base address per se. Host linear addresses that fall in a fence region are translated in their entirety by the tiling algorithm. It is as if the surface being tiled by the fence region has a base address in graphics memory equal to the fence base address, and all accesses of the surfaces are (possibly quite large) offsets from the fence base address. Fence regions have a virtual “left edge” aligned with the fence base address, and a “right edge” that results from adding the fence pitch to the “left edge”. Surfaces in the fence region must not straddle these boundaries.

Base addresses of surfaces that are to be accessed both by an internal graphics client and by the host have the tightest restrictions. In order for the surface to be accessed without GTT re-mapping, the surface base address (as set in SURFACE_STATE) must be a “Tile Row Start Address” (TRSA). The first address in each tile row of the fence region is a Tile Row Start Address. The first TRSA is the fence base address. Each TRSA can be generated by adding an integral multiple of the row size to the fence base address. The row size is simply the fence pitch in tiles multiplied by 4KB (the size of a tile.)

Tiled Surface Placement



The pitch in SURFACE_STATE must be set equal to the pitch of the fence that will be used by the host to access the surface if the same GTT mapping will be used for each access. If the pitches differ, a different GTT mapping must be used to eliminate the "extra" tiles (4KB memory pages) that exist in the excess rows at the right side of the larger pitch. Obviously no part of the surface that will be accessed can lie in pages that exist only in one mapping but not the other. The new GTT mapping can be done manually by SW between the time the host writes the surface and the device reads it, or it can be accomplished by arranging for the client to use a different GTT than the host (the PPGTT -- see *Logical Memory Mapping* below).

The width of the surface (as set in SURFACE_STATE) must be less than or equal to both the surface pitch and the fence pitch in any scenario where a surface will be accessed by both the host and an internal graphics client. Changing the GTT mapping will not help if this restriction is violated.

Surface Access	Base Address	Pitch	Width	Tile "Walk"
Host only	No restriction	Integral multiple of tile size <= 256KB	Must be <= Fence Pitch	No restriction
Client only	4KB-aligned	Integral multiple of tile size <= 256KB	Must be <= Surface Pitch	Restrictions imposed by the client (see Per Stream Tile Format Support)
Host and Client, No GTT Remapping	Must be TRSA	Fence Pitch = Surface Pitch = integral multiple of tile size <= 256KB	Width <= Pitch	Surface Walk must meet client restriction, Fence Walk = Surface Walk
Host and Client, GTT Remapping	4KB-aligned for client (will be tile-aligned for host)	Both must be Integral multiple of tile size <=128KB, but not necessarily the same	Width <= Min(Surface Pitch, Fence Pitch)	Surface Walk must meet client restriction, Fence Walk = Surface Walk

Per-Stream Tile Format Support

<table>
<tr><th>MI Client</th><th>Tile Formats Supported</th></tr>
<tr><td>CPU Read/Write</td><td>All</td></tr>
<tr><td>Display/Overlay</td><td>Y-Major not supported.
X-Major required for Async Flips</td></tr>
<tr><td>Blt</td><td>Linear and X-Major only
No Y-Major support</td></tr>
<tr><td>3D Sampler</td><td>All Combinations of TileY, TileX and Linear are supported. TileY is the fastest, Linear is the slowest.</td></tr>
<tr><td>3D Color,Depth</td><td>
<table>
<tr><th>Rendering Mode
Color-vs-Depth bpp</th><th>Buffer Tiling Supported</th></tr>
<tr><td>Classical
Same Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
<tr><td>Classical
Mixed Bpp</td><td>Both Linear
Both TileX
Both TileY
Linear & TileX
Linear & TileY
TileX & TileY</td></tr>
</table>
</td></tr>
</table>

Main Memory

The integrated graphics device is capable of using 4KB pages of physical main (system) memory for graphics functions. Some of this main memory can be “stolen” from the top of system memory during initialization (e.g., for a VGA buffer). However, most graphics operands are dynamically allocated to satisfy application demands. To this end the graphics driver will frequently need to allocate locked-down (i.e., non-swappable) physical system memory pages – typically from a cacheable non-paged pool. The locked pages required to back large surfaces are typically non-contiguous. Therefore a means to support “logically-contiguous” surfaces backed by discontiguous physical pages is required. The Graphics Translation Table (GTT) that was described in previous sections provides the means.

Optimizing Main Memory Allocation

This section includes information for software developers on how to allocate SDRAM Main Memory (SM) for optimal performance in certain configurations. The general idea is that these memories are divided into some number of page types, and careful arrangement of page types both within and between surfaces (e.g., between color and depth surfaces) will result in fewer page crossings and therefore yield somewhat higher performance.

The algorithm for allocating physical SDRAM Main Memory pages to logical graphics surfaces is somewhat complicated by (1) permutations of memory device technologies (which determine page sizes and therefore the number of pages per device row), (2) memory device row population options, and (3) limitations on the allocation of physical memory (as imposed by the OS).

However, the theory to optimize allocation by limiting page crossing penalties is simple: (a) switching between open pages is optimal (again, the pages do not need to be sequential), (b) switching between memory device rows does not in itself incur a penalty, and (c) switching between pages within a particular bank of a row incurs a page miss and should therefore be avoided.

Application of the Theory (Page Coloring)

This section provides some scenarios of how Main Memory page allocation can be optimized.

3D Color and Depth Buffers

Here we want to minimize the impact of page crossings (a) between corresponding pages (1-4 tiles) in the Color and Depth buffers, and (b) when moving from a page to a neighboring page within a Color or Depth buffer. Therefore corresponding pages in the Color and Depth Buffers, and adjacent pages within a Color or Depth Buffer should be mapped to different page types (where a page's "type" or "color" refers to the row and bank it's in).

Media/Video

The Y surfaces can be allocated using 4 page types in a similar fashion to the Color Buffer diagram. The U and V surfaces would split the same 4 page types as used in the Y surface.

Physical Graphics Address Types

The Physical Memory Address Types table lists the various *physical* address types supported by the integrated graphics device. Physical Graphics Addresses are either generated by Logical Memory mappings or are directly specified by graphics device functions. *These physical addresses are not subject to tiling or GTT re-mappings*.

Physical Memory Address Types

Address Type	Description	Range
MM_Address	Main Memory Address. Offset into physical, *unsnooped* Main Memory.	[0,TopOfMemory-1]
SM_Address	System Memory Address. Accesses are snooped in processor cache, allowing shared graphics/ processor access to (locked) cacheable memory data.	[0,512GB]

Graphics Translation Tables

The Graphics Translation Tables GTT (Graphics Translation Table, sometimes known as the global GTT) and PPGTT (Per-Process Graphics Translation Table) are memory-resident page tables containing an array of DWord Page Translation Entries (PTEs) used in mapping logical Graphics Memory addresses to physical memory addresses, and sometimes snooped system memory “PCI” addresses.

The base address (MM offset) of the GTT and the PPGTT are programmed via the PGTBL_CTL and PGTBL_CTL2 MI registers, respectively. The translation table base addresses must be 4KB aligned. The GTT size can be either 128KB, 256KB, or 512KB (mapping to 128MB, 256MB, and 512MB aperture sizes respectively) and is physically contiguous. The global GTT should only be programmed via the range defined by GTTMMADR. The PPGTT is programmed directly in memory. The per-process GTT (PPGTT) size is controlled by the PGTBL_CTL2 register. The PPGTT can, in addition to the above sizes, also be 64KB in size (corresponding to a 64MB aperture). Refer to the GTT Range chapter for a bit definition of the PTE entries.

Virtual Memory

This section describes the different paging models, their behaviors, and the page table formats.

GFX Page Tables

GPU supports three page table mechanisms

- PPGTT – per process GTT (private GFX)
- GGTT- global GTT

All page tables have the same PTE format, the difference was how to reach the final physical page and which fields with PTE are used.

Page Table Modes

The GFX Aperture and Display accesses are always mapped thru Global GTT. This is done to keep the walk simple (i.e. 1-level), however GT accesses to memory can be mapped via Global GTT and/or ppGTT with various addressing modes.

The walk modes are listed as following:

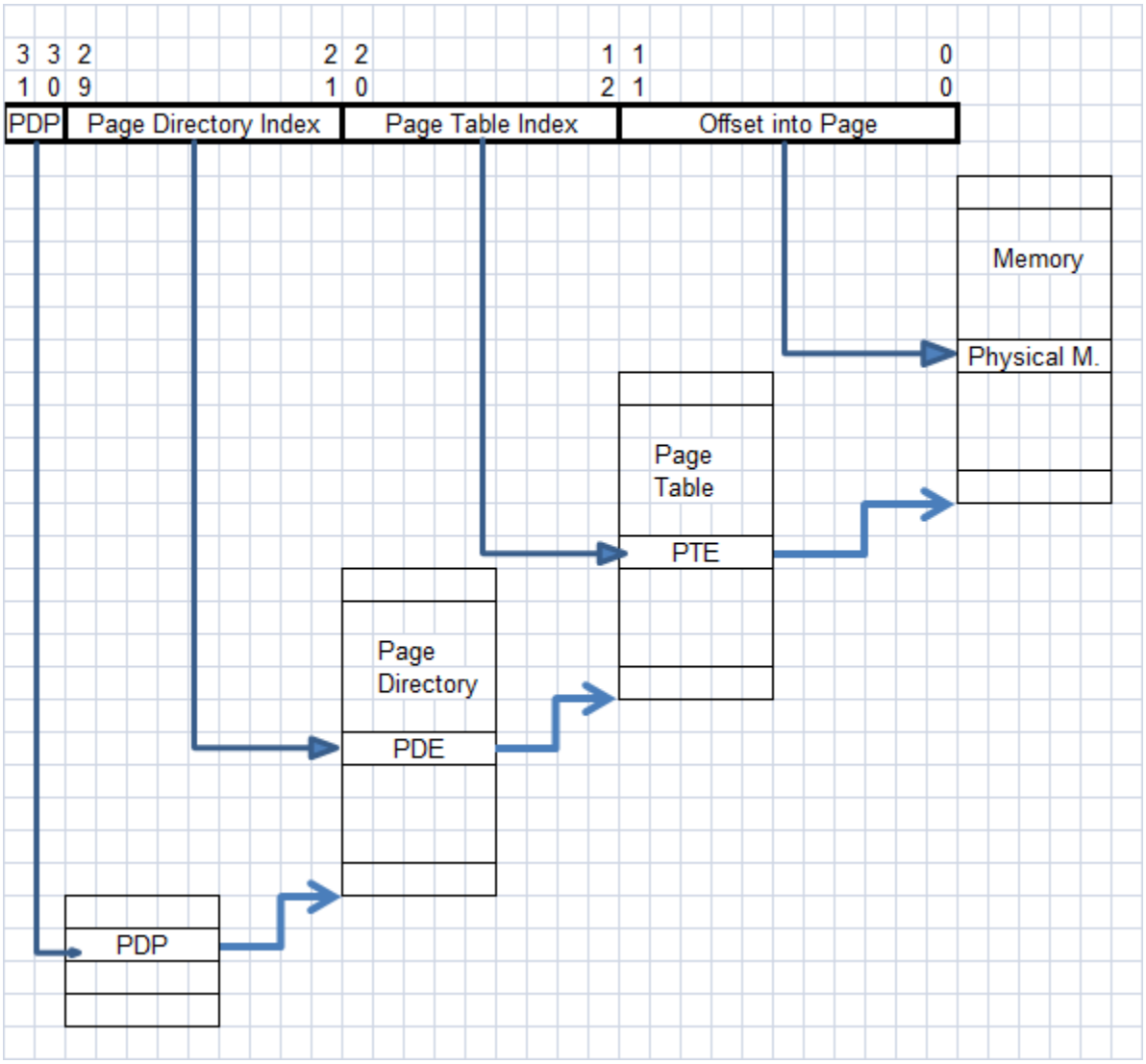
1. **Global GTT with 32b virtual addressing**: Global GTT usage is similar to pre-CHV, BSW behavior with extended capability to increase the VA to 4GB (from 2GB) and use a similar 64b PTE as ppGTT. The breakdown of the PTE for global GTT is given in later sections but fundamentally allows 1-level pagewalk where the 20b index is used to select the 64b PTE from stolen memory.
2. **Legacy 32b VA with ppGTT**: This is a mode where ppGTT page tables are managed via GFX s/w (driver) and context is tagged as Legacy 32b VA. Given each page walk is managed via 9b of the virtual address, 20b index is broken into 3 parts. However to optimize the walks and make it look like pre-CHV, BSW, s/w provides 4 pointers to page tables (called 4 PDP entries) – GPA. GFX h/w uses the four pointers and fetches the 4x4KB into h/w (for render and media) before the context execution starts. The optimization limits the dynamic (on demand) page walks to 1-level only.

Per Process GTT

Per process GTT mechanism has multiple hooks and mechanisms for s/w to prepare the page walks on hardware. The listed mechanisms here are selectable per-context and descriptors are delivered to hardware as part of context descriptor.

PPGTT for 32b Virtual Address

This page walk mechanism will be used for traditional 3D, Media type context. There is going to be a descriptor in the context header which will define the per process GTT walk that is required. For the standard context with 32bit virtual addressing, there is a possibility to take short cuts to reduce the overhead of the walk.



With 32-bit addressing the only entries that are needed for page directory pointers are 4x64bit locations (PDPE). For any standard context scheduling, it is required for s/w to provide 4 PDPEs as part of the context which would prevent h/w to do additional walks.

Hardware will do the remaining walks for PD and PTE similar to legacy behavior. In order to reduce the overhead of walks, hardware implements large caches for PDs.

Hardware does the remaining walks for PD and PTE similar to legacy behavior. To reduce the overhead of walks, hardware implements large caches for PDs:

- 4x4KB for 3D context
- 2x4x4KB for Media Context
- 4KB for VEBOX
- 4KB for Blitter

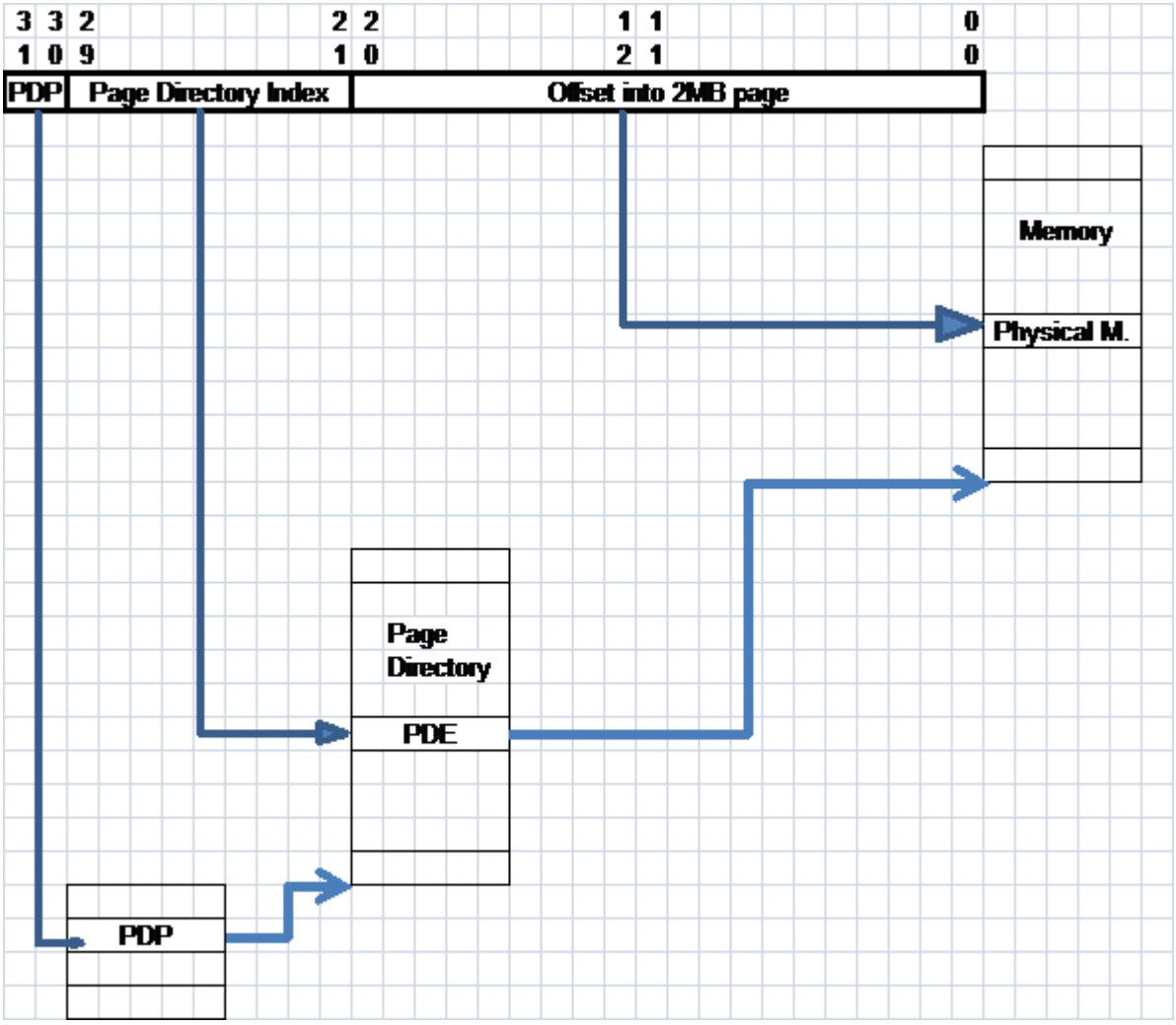
For Media and 3D context, the 16KB caches are preloaded for the entire page directory set up which limits the walk to 1-level before the final access. For remaining clients the PD cache is loaded on demand and can contain up to 512 entries.

Walk with 64KB Page

64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In page table every 16th entry (PTE#0, PTE#16, PTE#32….PTE#496) should be used to index. This is calculated using address[21:16]& “0000”. Note that hardware should not make any assumptions for any other PTEs.

Walk with 2MB Page

There is an option in the page walk to work with bigger page sizes, one of those sizes is 2MB pages. If allocated the page directory entry will indicate the page size and walk can be shortened as follows:

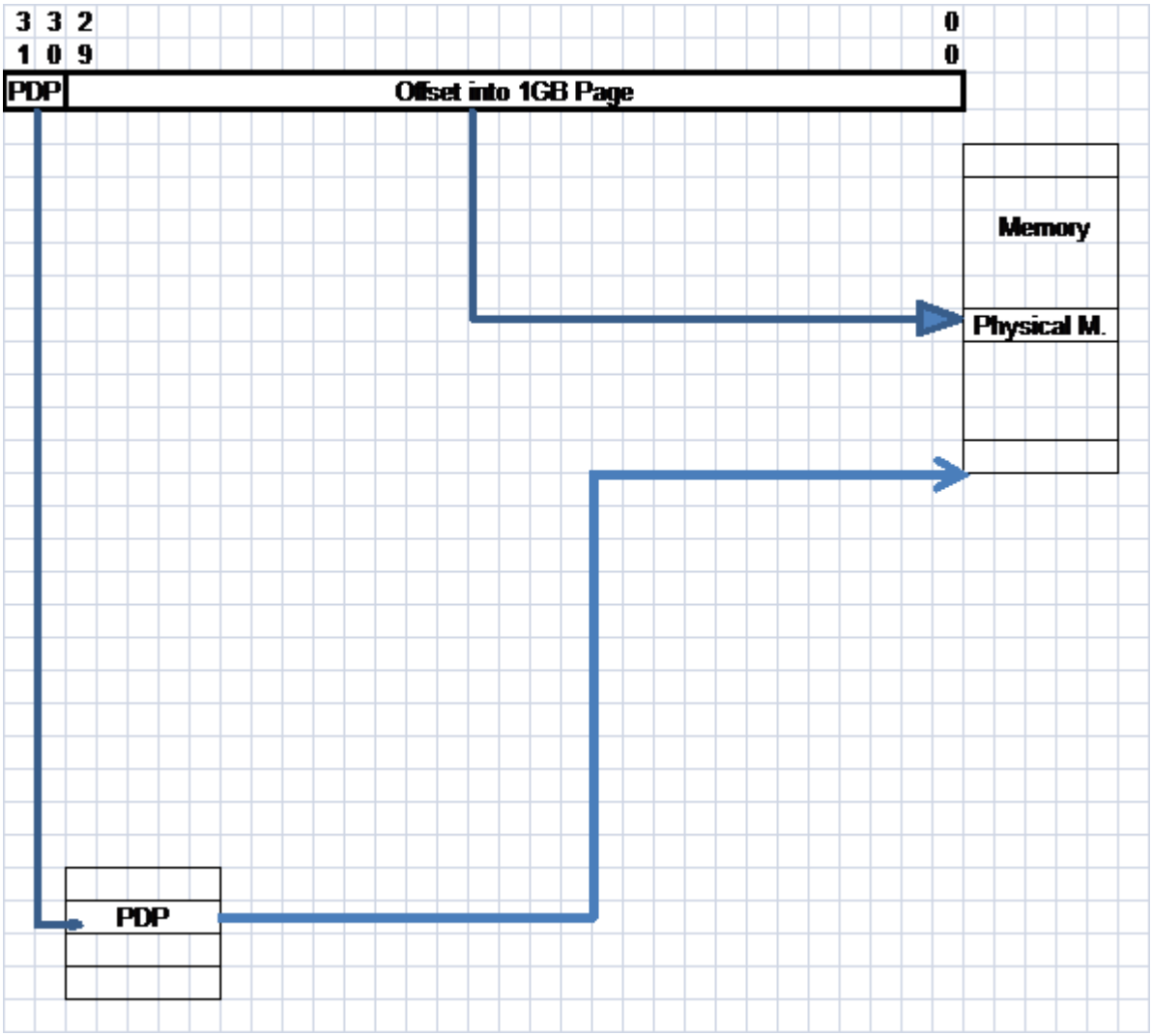


In this case there is no need to walk the page table after directory. And page directory has a pointer to 2MB range is physical memory.

Programming Note	
Context:	Walk with 2MB Page.
PPGTT32 is not going to support 2MB pages.	

Walk with 1GB Page

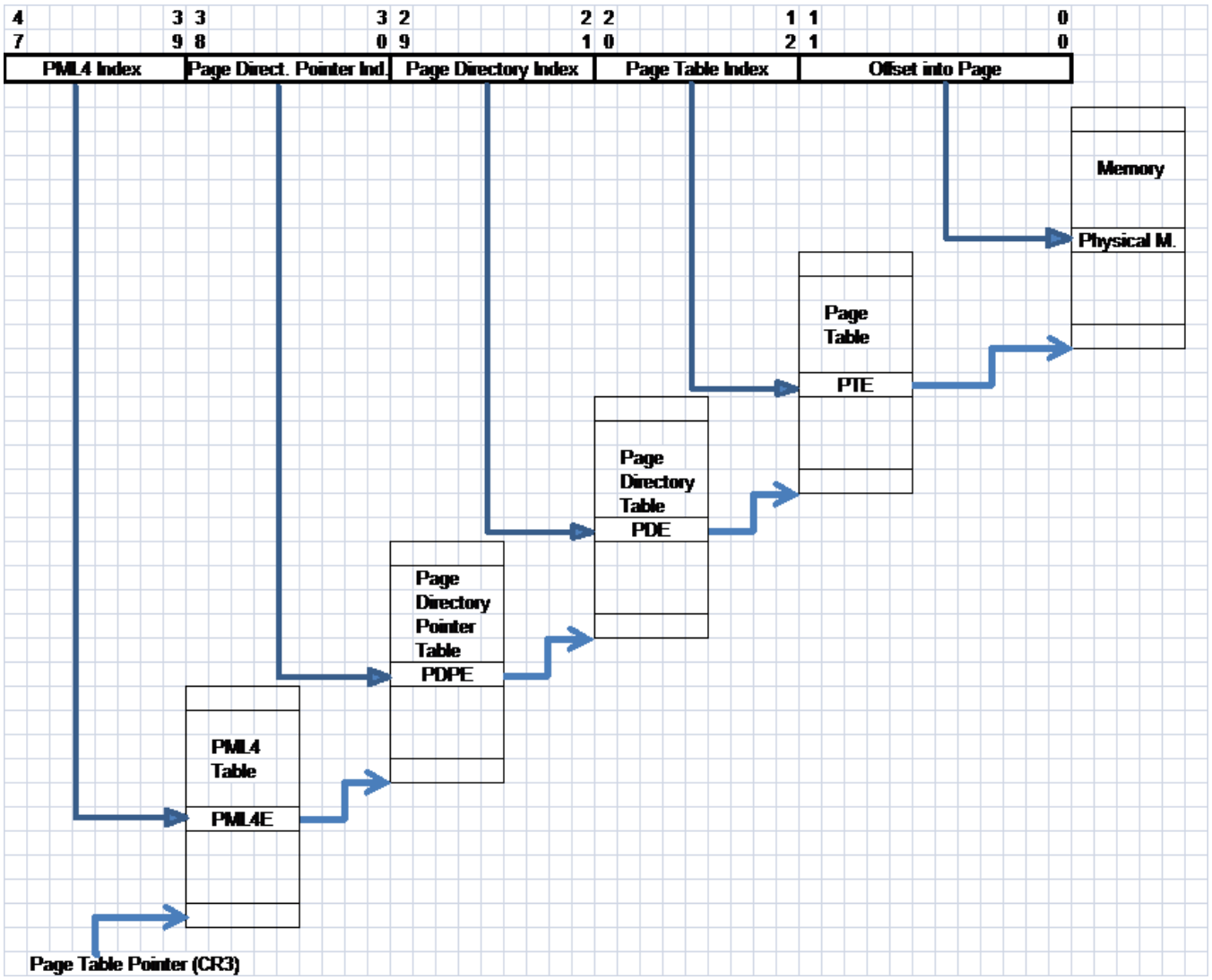
The same page walk is possible with 1GB page support as well.



Programming Note	
Context:	Walk with 1BG Page.
PPGTT32 is not going to support 1GB pages	

PPGTT for Standard Context (64b VA)

For advanced virtual addressing with legacy context, the full page walk mechanism needs to be exercised based on 48-bit canonical addressing.



64-bit (48b canonical) address requires 4-levels of page table format where the context carries a pointer to highest level page table (PML4 pointer or CR3). The rest of the walk is normal page walk thru various levels.

To repurpose the caches the following mechanism will be used:

- 3D: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- Media: 4KB to store PML4, 4KB as PDP cache, 2x4PD cache.
- VEBOX, Blitter: each with 4KB acting as PML4, PDP, PD cache.

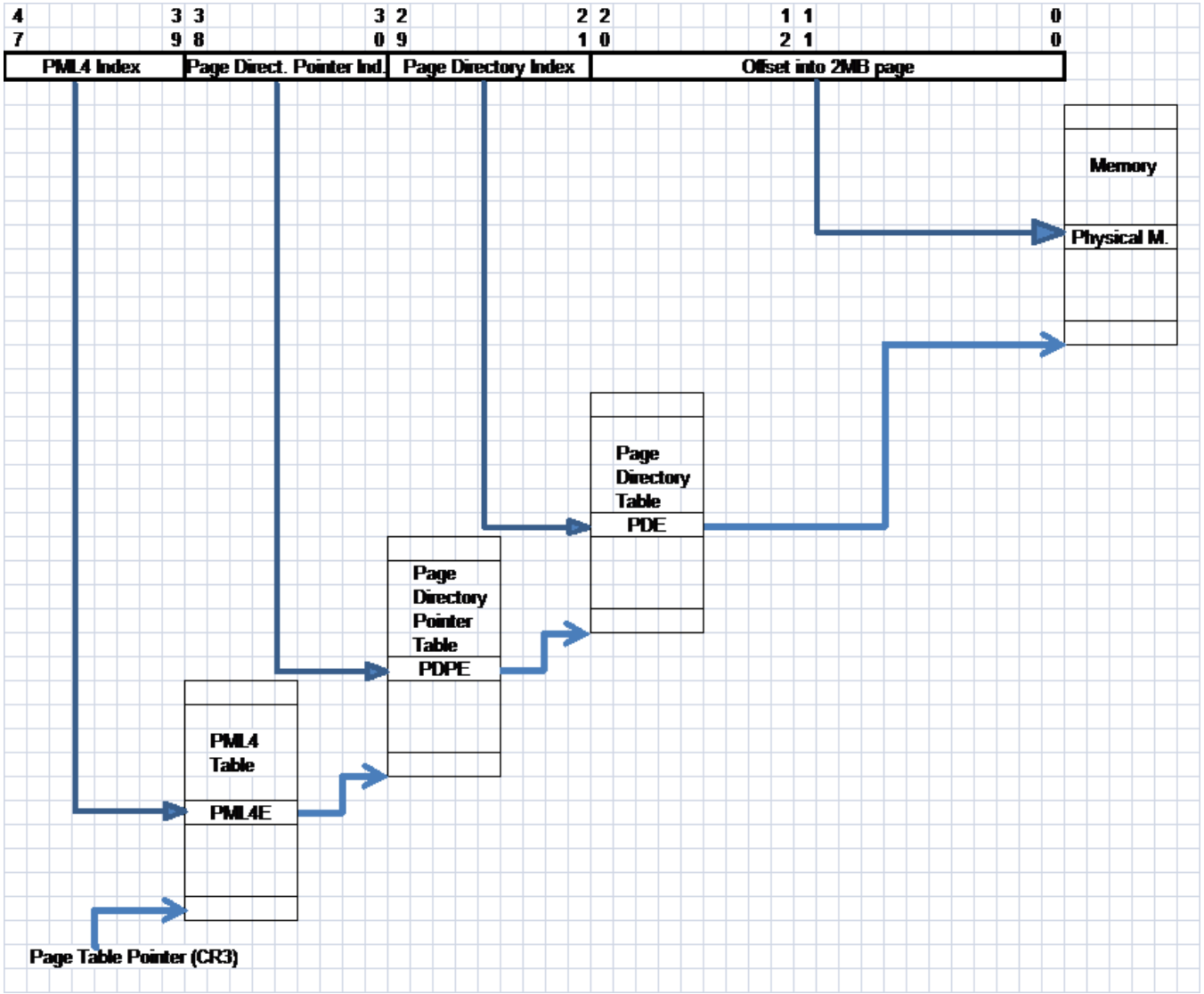
Programming Note	
Context:	PPGTT for Standard Context (64b VA)
Design can section the 512 entries within 4KB to separate areas for PML4, PDP, and PD.	

Walk with 64KB Page

64KB Page size has a slightly different usage for how PTEs are selected for the corresponding 64KB page. In page table every 16th entry (PTE#0, PTE#16, PTE#32….PTE#496) should be used to index. This is calculated using address[21:16]& "0000". Note that hardware should not make any assumptions for any other PTEs.

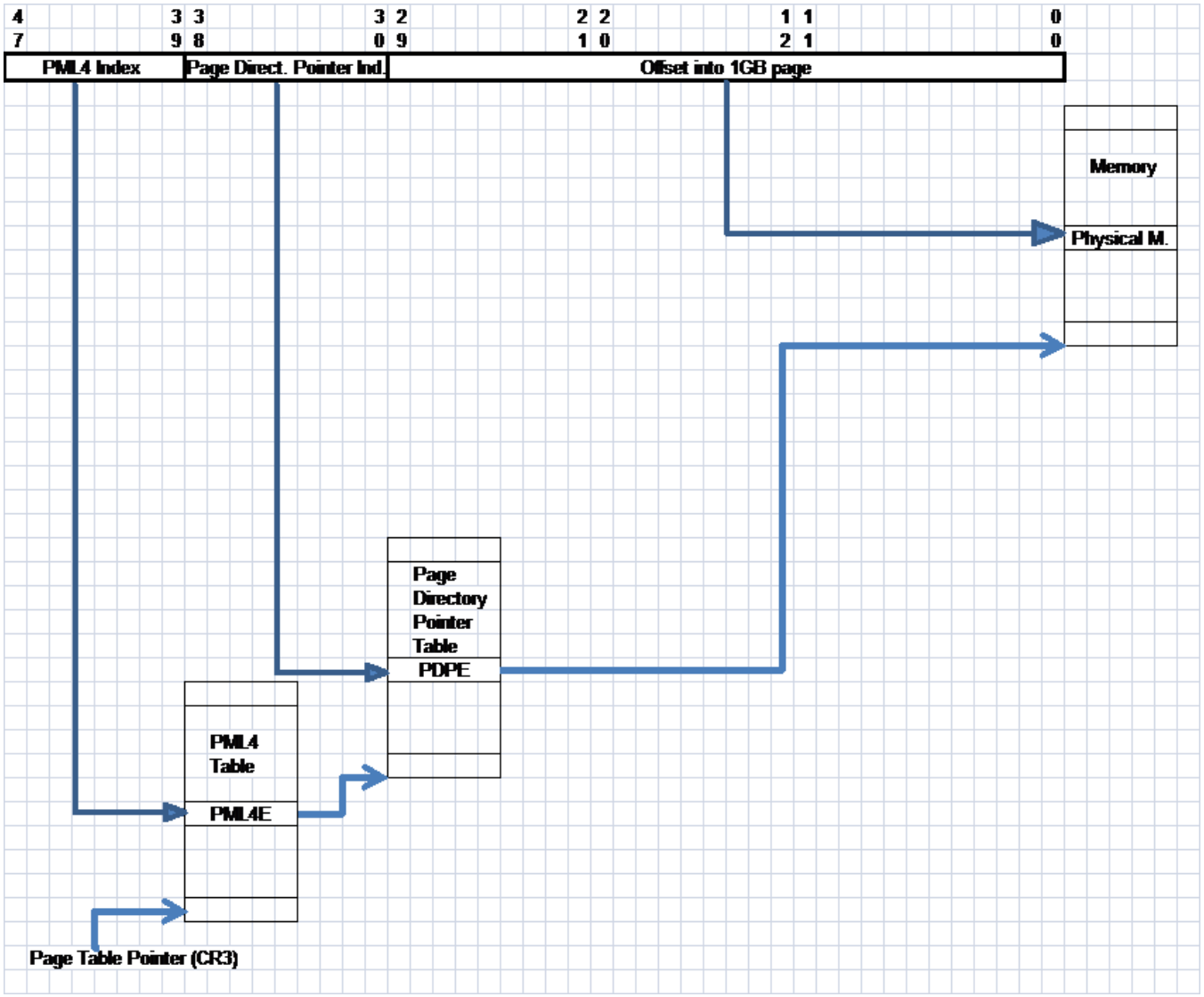
Walk with 2MB Page

Similar to the 32b VA walk, there is a support for larger pages where one of the sizes supported is 2MB.



Walk with 1GB Page

For the support for 1GB page size, the following mechanism is needed.



Programming Note	
Context:	Walk with 1 GB Page
PPGTT32 is not going to support 1GB pages.	

Global GTT

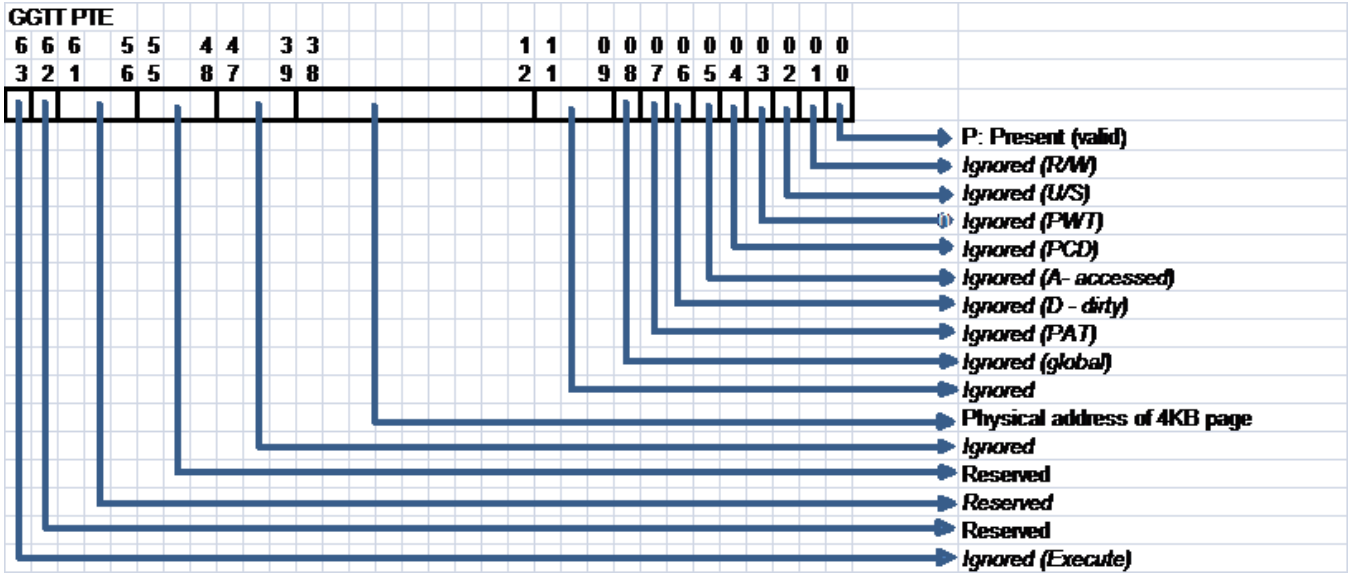
The Global GTT mechanism in CHV, BSW looks very similar to pre-CHV, BSW with the distinction of page table entry. Aperture and display will still use the global GTT even if GT core is mapped via per-process GTT.

The PTE format for CHV, BSW is updated to match per process GTT definitions and GSM is now expanded in size (2MB=>8MB) to cover for the entire 4GB (32b virtual addressing) space. Each entry corresponding to a 4KB page with 2^20 entries in GSM (each with 8B content)

For "*MI_update_GTT*", the page address provided 31:12 need to be shifted down to 22:3 for the correct QW position within the GGTT.

Page Table Entry

The following page table entry will be used for Global GTT:



- Present (Valid): The pointed PTE is valid
- *Ignored* - R/W (Read/Write): Are writes allowed to the region defined by this 4KB page. For GFX, in order 4KB memory to be usable it has to be both present and should also be write-able.
- *Ignored* - U/S (User/Supervisor access rights) : iGFX does not use these fields
- PWT/PCD/PAT bits are used as indexes into a PAT register which defines the cache attributes for the entire context.

PAT field is used to do the look up in private PAT for memory typing.

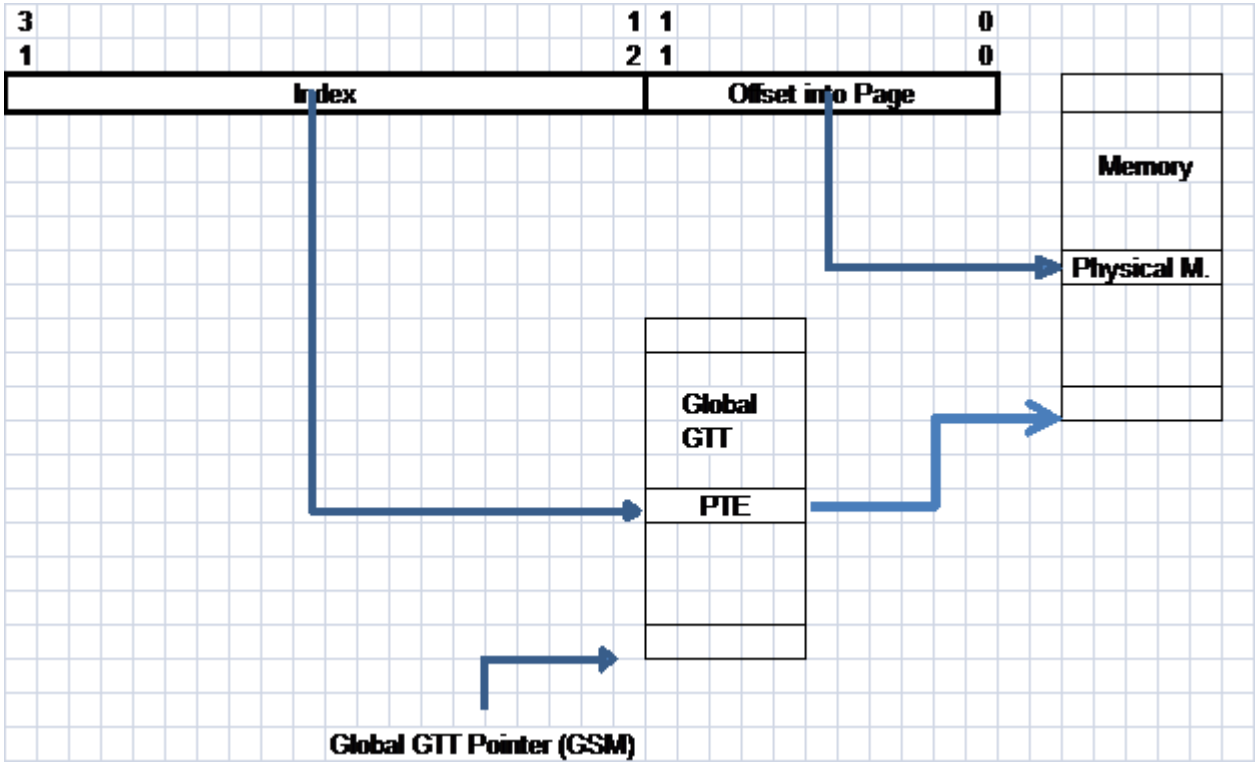
- *Ignored* - A (Accessed): It needs to be managed as the page table being accessed. Hardware needs to write this bit for the first access to the 4KB region defined with this PT entry.
- *Ignored* - D (Dirty): Hardware needs to set the dirty bit in page table if accessing this particular 4KB region in memory with the intention to modify it.
- *Ignored* - Global: this is not used by iGFX hardware, the field is used to identify global context where invalidation may not be required.
- Physical address of 4KB page

For the treatment of the page bit0 AND bit1 defines the validity of the page, the rest of the information is not relevant for Aperture and Display usage.

GGTT table entries are always read as uncacheable.

Page Walk

The global GTT page walk is identical to what it was before CHV, BSW. The only difference would be that each entry is 8B (instead of 4B) hence the entry selection needs to be updated once the corresponding Page Table miss read is returned.



GTT Cache

Processor graphics page walker implements a GTT cache which holds the remaining entries that are read as a cacheline but not used for the immediate page walk. This is only applicable in case of leaf walks and not including the 2MB/1GB page sizes. When s/w enables the use of 2MB/1GB page sizes, it will have to disable the GTT cache in CHV, BSW.

GFX Page Walker (GAM)

GPU supports various engines behind the same page walker. These streams/contexts are identified Client level IDs which are carried via the arbitration pipeline. Page walker using look-up tables does the correct selection for the page tables in case of concurrent context are running at the same time.

There are two different types of page table types:

Global graphics translation table (GGTT) is a single common translation table used for all processes. There can be many Per-process graphics translation table (PPGTT). This requires an additional lookup for translation. The actual location is not accessible directly via software since they're both located in graphics stolen memory (see graphics memory interface chapter for more detail).

Virtual Memory Structure	Memory Location
Global (GGTT)	GSM Only
Per-Process (PPGTT) – private	2 to4-level, Page Tables anywhere
Per-Process (IA32e) – shared	4 levels, Page Tables anywhere

IA32e compatible PPGTT is added to CHV, BSW to enable SVM (shared virtual memory) functions.

Context Definition for GFX Page Walker

Page Walker blocks need details about the context to decide on what type of page tables will be used, what would be the error handling cases would be and many other details to operate. The information will be passed to Page Walker (GAM) by the respective command streamer/DMA.

GAM supports the following engines:

- Render
- Media (VDBox) x2
- Blit
- VEBOX x2

The following fields will be sent to GAM:

- Context type (4 bits)
    - **Legacy vs Advanced Context**: Defines the context type and qualifies the rest of the fields. Same field may mean something else between the *Legacy* vs *Advanced* context. There is no restriction for what type of context can run in either combination.
        - *Requests without address-space-identifier (Legacy Context)*: These are the normal memory requests from endpoint devices. These requests typically specify the type of access (read/write/atomics), targeted DMA address/size, and identity of the device originating the request.
        - *Requests with address-space-identifier (Advanced Context)*: These are memory requests with additional information identifying the targeted process address space from endpoint devices supporting virtual memory capabilities. Beyond attributes in normal requests, these requests specify the targeted process address space identifier (PASID), and extended attributes such as Execute-Requested (ER) flag (to indicate reads that are instruction fetches), and Privileged-mode-Requested (PR) flag (to distinguish user versus supervisor access). For details, refer to the Process Address Space ID (PASID) Capability in the PCI-Express specifications.

Note CHV, BSW LP only supports Legacy PPGTT with 32b virtual addressing.

    - **A/D Support Enable**: Access and Dirty bits are used when OS managing the page tables and has been added to IA32e compatible page walk. Context will define whether A/D bits need to be managed via GPU. (only applicable in Advanced Context)
    - **Privileged Context Support**: Enables GPU to be able to run a privileged context which will translate into page table accesses regardless of user vs supervisor privileges. (only applicable in Advanced Context).
    - **32b vs 48b VA Support**: Enables 48b VA in page tables for the page walks. The rest of the h/w is seamless to 32b vs 48b VA address walks, however GAM will do the check and properly align the page walk to address bits.
        - *Note: Only applicable in Legacy Context, Advanced context is always 48b*.
        - Note: CHV, BSW LP only supports 32b VA
- **Function Number**: 3 bit field that defines the function number of the device. GFX device is always on BUS=0 and DEVICE=2. If we are not virtualized, our FUNCTION#=0 however if virtualized function number can be any 8 possible values (i.e. 0-7). The BUS/DEVICE/FUNCTION numbers are used to the initial walk for ROOT and CONTEXT tables.

- **PASID** – Process Address Space IDentifier: Use to identify the context that is submitted to h/w. We use the PASID in many places where during the page walk (i.e. PASID table look up) or while communicating with s/w on page faults. Each engine could be running an independent context with different PASID. The page walker should have a mechanism to be able to cache at least some number of PASID table entries (matching to the engine count) for faster walk.
- **Context ID** (Queue ID, Bell ID) – Context ID is used to further qualify the running context beyond the PASID. PASID is given per process, and same process may allocate multiple queues to communicate with h/w. The only way to further identify the process is to use an additional ID. For GFX h/w Context ID could be same as the bell number assigned to it. GAM h/w will use the context ID to populate the queue ID field while communicating page faults to s/w.
- **Page Table Pointers** – The field could be up to 256 bits (i.e. 4x64bits) to identify the page table pointers associated with the context. For legacy 32b context, the entire 256b is valid representing the 4 PDPTR table entries. For 48b legacy context only the lower 64b is relevant pointing to base of PML4. In case of advanced context, PASID is given in the context definition.

Context Definition Delivery

Context Definition is supposed to be delivered from the corresponding command streamer to GAM and GAM has independent storage for each engine present. Context Definition will be given by *CS to GAM via a new message:

Message: "Context Available"

GAM prepares for new context, cleans up internal state and does the proper fencing. Most of these steps should have been performed when context switch request was done for the previous context, but added here for completeness.

Message: "Context Receive Ready"

GAM is ready for the context. *CS writes all new context values into the descriptor registers. To push all context descriptors CS sends the following message to GAM also indicating new context descriptor is downloaded.

Message: "Context Launched"

GAM does the context requirements and sends the following message to CS to resume its command parser.

Message: Context Confirmed

GAM should send context confirmed message only after PD restore is done. CS waiting for context confirmed message will be treated as PD restore busy. Since all clients memory interface are blocked during PD restore it doesn't make any difference if the context confirmed message is send by GAM immediately or after PD restore.

Element Descriptor Register

General Description	Element Information: The register is populated by command streamer and consumed by GAM
Register Offset	See per engine list below

Bits	Access	Default	Field
63:32	RO	Xh	**Context ID:** Context identification number assigned to separate this context from others. Context IDs needs to be recycled in such a way that there could not be two active context with the same ID. This is a unique identification number by which a context is identified and referenced
31:12	RO	Xh	**LRCA:** Command Streamer Only
11:9	RO	Xh	**Function Number:** GFX device is considered to be on Bus0 with device number of 2. Function number is normally assigned as “0” however for gfx virtualization; there would be different function numbers which needs to be attached to context. Not used in CHV, BSW.
8	RO	Xh	**Privileged Context / GGTT vs PPGTT mode:** Differs in legacy vs advanced context modes: **In Legacy Context**: Defines the page tables to be used. This is how page walker come to know PPGTT vs GGTT selection for the entire context. “0”: Use Global GTT “1”: Use Per-Process GTT **In Advanced Context**: Defines the privilege level for the context “0”: user mode context “1”: supervisor mode context.
7:6	RO	Xh	**Fault Model:** “00”: Fault & Hang (chicken bit to survive). Same mode as gen7.5

Bits	Access	Default	Field
			"01": Fault & Halt/Wait. Same mode as gen7.5 "10": Fault & Stream & Switch "11": Fault & Continue: does not generate a page request to IOMMU.
5	RO	Xh	**Deeper IA coherency Support:** **In Advanced Context:** Defines the level of IA coherency "0": IA coherency is provided at LLC level for all streams of GPU (i.e. gen7.5 like mode) "1": IA coherency is provided at L3 level for EU data accesses of GPU
4	RO	Xh	**A&D Support / 32&64b Address Support:** Differs in legacy vs advanced context modes: **In Legacy Context**: Defines 32b vs 64b (48b canonical) addressing format "0": 32b addressing format "1": 64b (48b canonical) addressing format **In Advanced Context**: Defines A&D bit support "0": A&D bit management in page tables is NOT supported "1": A&D bit management in page tables is supported.
3	RO	Xh	**Context Type: Legacy vs Advanced** Defines the context type. "0": Advanced Context: Defines the rest of the advanced capabilities (i.e. OS page table support, fault models…). Note that advanced context is not bounded to GPGPU. "1": Legacy Context: Defines the context as legacy mode which is similar to prior generations of CHV, BSW. *Note that: Bits [8:4] differs in functions when legacy vs advanced context modes are selected.*
2	RO	Xh	**FR:** Command streamer specific
1	RO	Xh	**Scheduling Mode:** "1": Indicates execlist mode of scheduling. "0": Indicates Ring Buffer mode of scheduling.
0	RO	Xh	**Valid:** Indicates that element descriptor is valid. If GAM is programmed with an invalid descriptor, it will continue but flag an error.

PDP0/PML4/PASID Descriptor Register

General Description	PDP0/PML4/PASID: The register is populated by command streamer and consumed by GAM. It contains one of the 3 values which is determined by looking at the element descriptor.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:0	RO	Xh	**PDP0/PML4/PASID:** This register can contain three values which depend on the element descriptor definition. **PASID[19:0]**: Populated in the first 20bits of the register and selected when Advanced Context flag is set. **PML4[38:12]:** Pointer to base address of PML4 and selected when Legacy Context flag is set and 64b address support is selected **PDP0[38:12]:** Pointer to one of the four page directory pointer (lowest) and defines the first 0-1GB of memory mapping *Note: This is a guest physical address* *(unused bits need to be populated as 0's)*

PDP1 Descriptor Register

General Description	PDP1: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP1:** Pointer to one of the four page directory pointer (lowest+1) and defines the first 1-2GB of memory mapping *Note: This is a guest physical address* *(unused bits need to be populated as 0's)*

PDP2 Descriptor Register

General Description	PDP2: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP2:** Pointer to one of the four page directory pointer (lowest+2) and defines the first 2-3GB of memory mapping *Note: This is a guest physical address* *(unused bits need to be populated as 0's)*

PDP3 Descriptor Register

General Description	PDP3: The register is populated by command streamer and consumed by GAM. It contains one of the pointers to PD.
Register Offset	See per engine list below

Bits	Access	Default	Field
63:12	RO	Xh	**PDP3:** Pointer to one of the four page directory pointer (lowest+3) and defines the first 3-4GB of memory mapping *Note: This is a guest physical address* *(unused bits need to be populated as 0's)*

List of Registers and Command Streamers

The following registers are message registers and not written directly by software.

Engine	Offset	Description
Render	x4400h	Element Descriptor Register
	x4408h	PDP0/PML4/PASID Descriptor Register
	x4410h	PDP1 Descriptor Register
	x4418h	PDP2 Descriptor Register
	x4420h	PDP3 Descriptor Register
Media0 (VDBOX0)	x4440h	Element Descriptor Register
	x4448h	PDP0/PML4/PASID Descriptor Register
	x4450h	PDP1 Descriptor Register
	x4458h	PDP2 Descriptor Register
	x4460h	PDP3 Descriptor Register
Media1 (VDBOX1)	x4480h	Element Descriptor Register
	x4488h	PDP0/PML4/PASID Descriptor Register
	x4490h	PDP1 Descriptor Register
	x4498h	PDP2 Descriptor Register
	x44A0h	PDP3 Descriptor Register
VEBOX	x44C0h	Element Descriptor Register
	x44C8h	PDP0/PML4/PASID Descriptor Register
	x44D0h	PDP1 Descriptor Register
	x44D8h	PDP2 Descriptor Register
	x44E0h	PDP3 Descriptor Register
Blitter	x4500h	Element Descriptor Register
	x4508h	PDP0/PML4/PASID Descriptor Register
	x4510h	PDP1 Descriptor Register
	x4518h	PDP2 Descriptor Register
	x4520h	PDP3 Descriptor Register

Messages

Message Name	Source	Destination	Category	Address (Hex)	Bit	Mask Bit	Value	Description
Context Available	CS (GT)	GAM (GT)	self-clear	4004	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	CS(GT)	self-clear	3438	0	16	1	Signal ack from GAM to CS in response to Context Available message from CS to GAM.
Context Launched	CS (GT)	GAM (GT)	self-clear	4004	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	CS(GT)	self-clear	3438	1	17	1	Signal ack from GAM to CS in response to Context Launched message from CS to GAM.
Context Available	BCS (GT)	GAM (GT)	self-clear	4014	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	BCS(GT)	self-clear	23438	0	16	1	Signal ack from GAM to BCS in response to Context Available message from BCS to GAM.
Context Launched	BCS (GT)	GAM (GT)	self-clear	4014	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	BCS(GT)	self-clear	23438	1	17	1	Signal ack from GAM to BCS in response to Context Launched message from BCS to GAM.
Context Available	VECS (GT)	GAM (GT)	self-clear	4010	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	VECS(GT)	self-clear	1B438	0	16	1	Signal ack from GAM to VECS in response to Context Available message from VECS to GAM.
Context Launched	VECS (GT)	GAM (GT)	self-clear	4010	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VECS(GT)	self-clear	1B438	1	17	1	Signal ack from GAM to VECS in response to Context Launched message from VECS to GAM.
Context Available	VCS0 (GT)	GAM (GT)	self-clear	4008	0	16	1	Signal request from CS to GAM as new context is about to be submitted.

Message Name	Source	Destination	Category	Address (Hex)	Bit	Mask Bit	Value	Description
Context Receive Ready	GAM (GT)	VCS0(GT)	self-clear	13438	0	16	1	Signal ack from GAM to VCS in response to Context Available message from VCS to GAM.
Context Launched	VCS0 (GT)	GAM (GT)	self-clear	4008	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VCS0(GT)	self-clear	13438	1	17	1	Signal ack from GAM to VCS in response to Context Launched message from VCS to GAM.
Context Available	VCS1 (GT)	GAM (GT)	self-clear	400C	0	16	1	Signal request from CS to GAM as new context is about to be submitted.
Context Receive Ready	GAM (GT)	VCS1(GT)	self-clear	1D438	0	16	1	Signal ack from GAM to VCS in response to Context Available message from VCS to GAM.
Context Launched	VCS1 (GT)	GAM (GT)	self-clear	400C	1	17	1	Signal indicator to GAM that context descriptor is pushed.
Context Confirmed	GAM (GT)	VCS1(GT)	self-clear	1D438	1	17	1	Signal ack from GAM to VCS in response to Context Launched message from VCS to GAM.

Updating Page Table Pointers (aka PD Load)

In case of legacy context, driver is allowed to add/remove pages as long as it is ensured that h/w is not using these entries. Pre-CHV, BSW flow allowed a mid-context PD load to update the PD entries and directed h/w to reload updated entries. CHV, BSW legacy context will require a similar mechanism.

Instead of a PD load, the new mechanism will let the driver update the page table pointers via sending a reload command. Mechanism will be overlapped to same sort messaging between CS and GAM to deliver the context header. CS will send the following message to GAM:

Message: Context Reload

GAM will respond immediately to CS with the following message

Message: Context Confirmed

Meanwhile GAM will block the related interfaces and updates the PDs or PML4.

Page Walker (GAM) Reset

GAM gets all the engine specific resets as well as device and bus resets to manage its internal logic domains. It is the expectation of SW when a particular GPU engine (i.e. Render, Media, ...) gets reset, all its related HW is cleared and comes out fresh for reprogramming. That is true for most of the logic with the exception of some shared HW blocks. The following blocks require additional steps (post-reset) from SW to further clean-up the HW:

- **Hardware TLBs:** The caching structures for the page walks are often considered shared resources. The expectation is for GFX driver to clear the TLBs via "TLB Invalidate" prior to reusing the engine post reset. This is the same process that was followed on previous GPU generations.
- **Page Requests:** At the time of the reset HW may have outstanding page requests to SW for page faulted accesses. These requests could be at any level hence it is required for SW to clear these paging requests pre/post-engine reset. Engine reset ensures no new page requests would be sent from HW. Page requests could be at the "page request queue" in memory where they could be mapped to a dummy page post engine reset completion. Or they could be at the MMIO registers which will block the completion of the reset; it is up to SW to service paging request interrupts without waiting for the completion of reset request.

Device reset (FLR) covers most of the page walker, however there are exceptions where IOMMU structures and all messaging towards rest of the system (system agent) should not be impacted by it. All external interactions and IOMMU related blocks are kept under bus (system) reset. GAM will keep the following blocks outside the device reset:

- IOMMU registers and content.
- All system agent messaging structures (including translation enable flows, root pointer structures, DMA fault reporting pieces.).

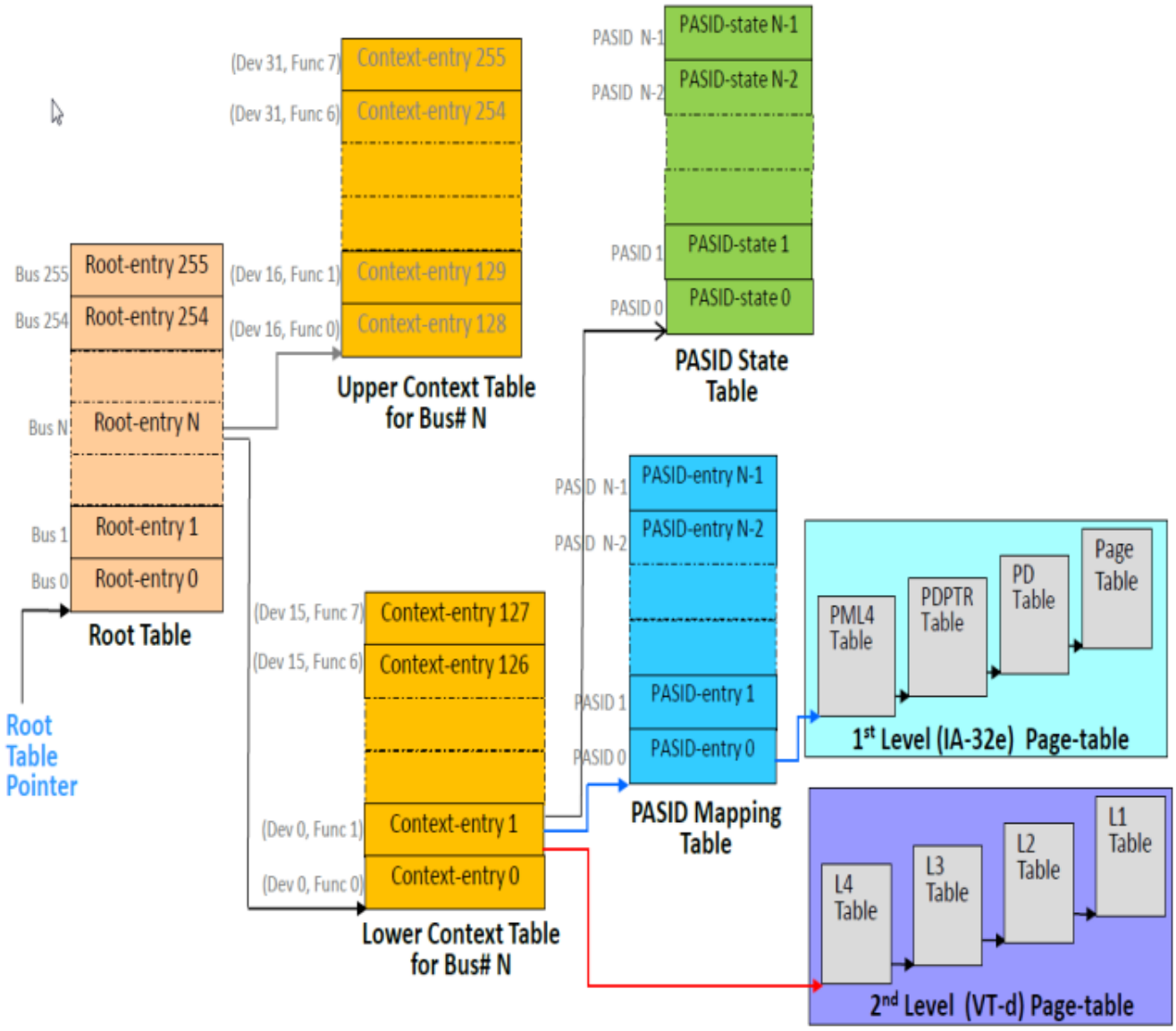
An engine being reset also means the particular context that engine is running, is complete or taken out. That will require GAM to decrement the PASID_State Counter if the engine was running a PASID based (advanced) context. For FLR (device reset), a similar requirement holds. In case of device reset, GAM would need to decrement all the PASID state counters that are active on the GPU before completing the sequence.

Legacy Context

Legacy context could use either Global GTT or Per Process GTT which is given to page walker as part of the context descriptor. Even under PPGTT, there could be accesses from Command Streamers that would require to use Global GTT which requires to treat the walk requirement per transaction.

For Legacy context indicator command streamer is going to pass the "context type" information along with other parameters that defines how certain behavior for paging needs to be.

Full Walk



In the full walk case (i.e. advanced context), the root of the 1st and 2nd level page tables share a common source. Both the root table and context table is walked with the assumption of GFX device is always on Bus#0 and it is always Device#2.

Function number however is part of the context and it can be non-Zero only for virtualized modes. GAM will receive the function number (for Context table look-up) as part of the context. Both Root entry and Context Entry should be fetched along with PASID Table entry prior to running the context accesses.

TLB Caching and Management

As compared to previous generation of TLB entry, IA32e page translation entry is quite different. At every stage of the page different bits need to be taken into account and proper treatment is required. Regardless of PPGTT vs GGTT usage, the paging entry has the same format. Linear address are translated using a hierarchy of in-memory paging structures located using the contents of CR3. IA-32e paging translates 48-bit linear addresses to 52-bit physical addresses. Although 52 bits corresponds to 4 PBytes, linear addresses are limited to 48 bits; at most 256 TBytes of linear-address space may be accessed at any given time. IA-32e paging may map linear addresses to 4-KByte pages, 2-MByte pages, or 1-GByte pages.

63	62–(M-1)	(M-1)–12	11–8	7	6	5	4	3	2	1	0	
Reserved[2]		Address of PML4 table		Ignored			PCD	PWT	Ign.			CR3
XD	Ignored	Rsvd.	Address of page-directory-pointer table		Ign.		Rsvd	Ign	A	PCD	PWT	U/S	R/W	1	PML4E: present
Ignored														0	PML4E: not present
XD	Ignored	Rsvd.	Address of 1GB page frame	Reserved		PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDPTE: 1GB page
XD	Ignored	Rsvd.	Address of page directory				Ign.		0	Ign	A	PCD	PWT	U/S	R/W	1	PDPTE: page directory
Ignored														0	PDTPE: not present
XD	Ignored	Rsvd.	Address of 2MB page frame	Reserved		PAT	Ign.	G	1	D	A	PCD	PWT	U/S	R/W	1	PDE: 2MB page
XD	Ignored	Rsvd.	Address of page table				Ign.		0	Ign	A	PCD	PWT	U/S	R/W	1	PDE: page table
Ignored														0	PDE: not present
XD	Ignored	Rsvd.	Address of 4KB page frame				Ign.	G	PAT	D	A	PCD	PWT	U/S	R/W	1	PTE: 4KB page
Ignored														0	PTE: not present

The following rules apply:

1. M is an abbreviation for MAXPHYSICAL ADDRESS.
2. Reserved fields must be "0".
3. Ignored field must be ignored. (There could be private information.)

All ignore options are part of the context entry and come from the IOMMU definition.

TLB Caches

For CHV, BSW the caching structures are separated as following with the architectural view, this is also applicable to s/w view of these caches when it comes to invalidations.

Context Cache - CC

This is the storage for context table entry which is achieved as part of root/context table walk.

Context cache can also be invalidated with directed invalidations, where HW needs to invalidate the content of the context cache along with all low level caches.

PASID Cache - PC

This is where the HW copy of the PASID table entry is kept and it is per context. This makes it unique for every HW engine that could be running an independent context (per GAM):

- Render/GPGPU
- MFX (VDBOX) – 1
- MFX (VDBOX) – 2
- Video Enhancement (VEBOX) – 1
- Video Enhancement (VEBOX) – 2
- Blitter

The cache content is updated if the corresponding engine is running an advanced context where its page table pointers are accessible via PASID table. In case of legacy context running engine, corresponding PASID Cache entry is not valid. Recommendation is to keep ONE physical storage per engine which is filled/invalidated during the context switch time.

PASID Cache can also be invalidated with the directed invalidations along with low level caches and needs to be re-filled prior to context resuming.

Intermediate Page Walk Caches (PML4, PDP, PD) – PWC

These are the stages where intermediate page walk entries are cached to speed-up/shorten the page walk when final TLB is missed. Each level can be cached separately or along with different levels, the cacheability structures will have programmability to move the boundary of different levels to accommodate more/less on each page walk level. However as a concept, for legacy 32b addressing mode, requirement is to cache 4PDPs along with 4x4KB PDs for certain engines, at least for render and media. The others will use cache concept.

TLB – Final Page Entry

The size of the TLBs has been increased over the previous generation and should be targeting the below table:

- L3 TLB: 768 TLB entries – This is where all HDC, I$, Constant, State, and Sampler streams are stored.
- MFX: 512 TLB entries – All Media streams (split 256/256 between two media engines).
- BLT: 32 entries.
- Z: 512 TLB entries – All depth accesses.
- C: 256 (256 TLB entries) – All color accesses.
- FF: 128 (128 TLB entries) – All FF accesses to memory.
- VLF: 32 (32 TLB entries) – Media surface.
- GAV: 192 (192 TLB entries) – Video enhancement. Increased compared to other Gen8 projects.
- WiDi: 64 (64 TLB entries) – Wireless Display.

All TLB entries are increased to 48b to contain larger address as well as the page attributes attached to it.

The max size of a single TLB is 256 entries, larger quantities have to be handled as set-associative storages. Set associativity is managed by low order page bits (i.e. address#12, address#13, ...).

TLB Entry Content

When a page walk entry is cached (or loaded prior to context start), certain bits need to be cached as well along with the physical address bits. The treatment on these bits would be considered when a HIT vs MISS decision needs to be made during a look up.

The purpose of caching is to accelerate the paging process by caching individual translations in **translation look-aside buffers** (**TLBs**). Each entry in a TLB is an individual translation. Each translation is referenced by a page number. It contains the following information from the paging-structure entries used to translate linear addresses with the page number:

- The physical address corresponding to the page number (the page frame).
- The access rights from the paging-structure entries used to translate linear addresses with the page number:
  - The logical-AND of the R/W flags.
  - The logical-AND of the U/S flags.
  - The logical-OR of the XD flags.
- Attributes from a paging-structure entry that identifies the final page frame for the page number (either a PTE or a paging-structure entry in which the PS flag is 1):
  - The dirty flag.
  - The memory type.

PRESENT: This is the same VALID bit description we had in previous page table designs. The lack of present bit (i.e. bit[0]=0) points that rest of the information in the page table entry is being invalid. For some fault models, even NOT PRESENT entries are cached to filter further page faults (*see fault models on caching page faulting entries*). If such entry is cached, there are couple ways that it can be removed from the page tables:

1. LRA selection where the entry becomes a victim for replacement
2. Global or Selective invalidation
3. Page fault response stating the faulting page is now fixed.

R/W Privilege: Certain pages can be allocated as read-only and write operations are not allowed. To make this check work, TLB has to keep the R/W bit. This bit has no affect on read operations; however for write operation privilege needs to be checked. If there is mis-match, the result of the TLB look-up should be a MISS. This does not mean a page fault immediately; the walk has to be re-done as for any TLB MISS result. There are cases OS may change page table privileges without invalidating pages in TLB (*note: all downgrades result in invalidation of the TLB, however upgrades can be done silently hence re-walk is required*). In case where the TLB Miss is due to privilege mis-match, the existing entry from TLB has to be invalidated and page walk will bring in the most up-to-date copy from memory.

The R/W privilege on final frame is generated as a logical-AND process of all upper page walks pointing to this location.

User vs Supervisor Privilege: The GPU typically operates in user mode when it comes to page tables. So the GTT walk can be treated as faulted when GPU encounters a page with supervisor privileges and the context is marked as user mode. The faulted entry can be cached back into TLB with "P" bit off indicating a faulted entry. However the page fault report should carry the correct reason why h/w detected the fault in the first place which was the user vs supervisor privilege. There is an option in context header to define the context as supervisor, than it legal to access supervisor pages.

- This is not stored in TLB

The U/S privilege on final frame is generated as a logical-AND process of all upper page walks pointing to this location.

Accessed Bit: This where a stage of the page walk cannot be used if the accessed bit is not set for that level in the page walk. This is true for both storage into TLB as well as to make progress on the page walk. In order to achieve the process of Accessed bit, every stage of the ppGTT read is done via a new semantics between the GAM and GTI such that GTI can atomically process A-bit w/o running into access violations. The details of the semantics are defined as part of the following sections. The "A" bit does not need to be stored as part of the TLB, just the fact that a valid page table entry is present in the TLB does mean that h/w took care off the "A" bit at the time the page was brought up to TLB. Note that TLB prefetching is disabled when A-bit management is enabled.

IA32e mode page tables cannot co-exist with TLB pre-fetching due to lack of A-bit management for all entries of the line.

- This is not stored in TLB

Dirty Bit: Similar to accessed bit, dirty bit needs to be managed. It is only applicable for "write" accesses. Given there are cases where a TLB entry was acquired as part of a read operation, the presence of D-bit should be maintained with the TLB. This gives us the capability to declare a TLB miss for a write access when the D-bit is not set even though TLB has a valid translation. In such case, The TLB entry needs to invalidated and the final stage of the walk needs to be re-done to ensure most up-to-date copy of GTT entry is brought into h/w. The operation of Dirty bit update is also atomic similar to A-bit management.

Execute (XD) Bit: XD bit is also present on every stage of the walk and applicable to executable code that GT would be fetching. In the first pass, instruction cache accesses are not allowed to proceed if the corresponding page does not have the execute credentials set properly. Similar treatment of the TLB entry as privilege bits is expected. A page entry that was already cached in TLB and later accessed for instruction space will have to check the XD bit which is also stored in TLB. If mis-match, the end result is a TLB miss and walk has to be re-done replacing the different stages of the walk.

The XD privilege on final frame is generated as a logical-OR process of all upper page walks pointing to this location.

Faulted Bit: There are usage models where the faulted entries are cached in TLB. This is to filter further faults to the same page as opportunistic way to prevent fault storms. When faulted bit is set the address is included in the TLB look up but final treatment is fault filtering. The rest of the bits are used to define what would be the reason for the fault. If the look-up conflicts with the original faulted reason, a re-walk is required. As a basic case, take a read access bringing up a PTE with W-flag cleared. A

subsequent write access has a conflict on privilege, and it will perform a re-walk. If the result of the re-walk is W-flag set, than TLB is upgraded and write makes progress. However if the result is still W-flag cleared, the write access will fault and TLB entry will be tagged as a faulted entry with only read-allowed. Subsequent write accesses will be filtered as fault but read accesses should cause a re-walk of the page and if successful, the TLB can be updated with PTE as valid with read-only attribute.

TLB Accessed and Dirty Flags

For any paging-structure entry that is used during linear-address translation, bit 5 is the **accessed** flag. For paging-structure entries that map a page (as opposed to referencing another paging structure), bit 6 is the **dirty** flag. These flags are provided for use by memory-management software to manage the transfer of pages and paging structures into and out of physical memory.

Whenever the processor and/or GPU uses a paging-structure entry as part of linear-address translation, it sets the accessed flag in that entry (if it is not already set).

Whenever there is a write to a linear address, the processor and/or GPU sets the dirty flag (if it is not already set) in the paging-structure entry that identifies the final physical address for the linear address (either a PTE or a paging-structure entry in which the PS flag is 1).

Memory-management software may clear these flags when a page or a paging structure is initially loaded into physical memory. These flags are “sticky,” meaning that, once set, the processor and/or GPU does not clear them; only software can clear them.

A processor and/or GPU may cache information from the paging-structure entries in TLBs and paging-structure caches (see Section 4.10). This fact implies that, if software changes an accessed flag or a dirty flag from 1 to 0, the GPU might not set the corresponding bit in memory on a subsequent access using an affected linear address

Accessed bit is applicable to every stage of the page walk, however the dirty bit is only applicable to final stage of the walk.

The rule states that a particular access cannot be committed until the Accessed and/or Dirty bits are not visible to page management s/w. In order for GPU to follow the rule, GTT accesses (when A/D bits are supported) are going to be done via a special cycle definition between GAM and GTI.

Updating A/D Bits

New atomic operations are added to GAM to GPU interface (GTI) to handle paging entries. GAM has to set the correct atomic opcodes based on the access type and context entry controls as well as level of access.

Requires setting for opcodes are given in the table below. The steps of operations in the atomic ALUs are given later in the document.

The Following Atomics are only applicable in GTI and used for Page Walks

R/W => Bit[0]

Extended Access required => Bit[1]

Write Protect Enable => Bit[2]

Intermediate Entry => Bit[3]

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
Atomic_Page_update_0000	1100_0000	Read Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0001	1100_0001	Write Access Extended Access bit is disabled Write Protection is disabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0000	1100_0010	Read Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0001	1100_0011	Write Access Extended Access bit is enabled Write Protection is disabled Final PTE	Set bit[5,6,10] if not set		new_dst

Atomic_Page_update_0100	1100_0100	Read Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5] if not set		new_dst
Atomic_Page_update_0101	1100_0101	Write Access Extended Access bit is disabled Write Protection is enabled Final PTE	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0100	1100_0110	Read Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0101	1100_0111	Write Access Extended Access bit is enabled Write Protection is enabled Final PTE	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_0000	1100_1000	Read Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5] if not set		new_dst

Atomic_Page_update_0001	1100_1001	Write Access Extended Access bit is disabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst
Atomic_Page_update_0000	1100_1010	Read Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0001	1100_1011	Write Access Extended Access bit is enabled Write Protection is disabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst
Atomic_Page_update_0100	1100_1100	Read Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5] if not set		new_dst
Atomic_Page_update_0101	1100_1101	Write Access Extended Access bit is disabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6] if not set		new_dst

Atomic_Page_update_0100	1100_1110	Read Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,10] if not set		new_dst
Atomic_Page_update_0101	1100_1111	Write Access Extended Access bit is enabled Write Protection is enabled Intermediate Paging Entry	Set bit[5,6,10] if not set		new_dst

Atomic updates are only possible for cacheable memory types. There could be cases where the PTE could be in WT/WC/UC space where atomic update is not possible via WB space. Those are the cases where IA cores use bus lock to update the A/D bits in PTE.

GT core is not capable of supporting bus locks and has the following options. These options will be enabled/disabled via register space.

Option#1: Ignore the PAT/MTRR setting of the PTE and update the space as WB with atomic ops. This is the place GAM will decide to go forward with atomic updates assuming WB space works

Option#2: Once the memory type is determined and the end result of the page is WC/UC/WT space, we can not guarantee an atomic update. GAM will report an application error (catastrophic) to the scheduler and handle the case as error.

Bit	Access	Default Value	Description
1	R/W	0b	**A/D Bit Update on non-WB Space:** A/D bit updates are only possible via atomic operations which are required to be on WB space to work properly. On non-WB spaces, the A/D bit updates are done via bus locks which are not supported for GT. **"1":** Ignore the page level cacheability and do atomic updates for A/D bit management **"0":** Detect the page level cacheability as part of the atomic operation and throw a catastrophic error when non-WB space is seen for A/D bit updates.

PAT (IA32e)

If the PAT is supported, paging contributes to memory typing in conjunction with the PAT and the memory-type range registers (MTRRs) as specified.

The PAT is a 64-bit data structure defined in context entry when advanced context is chosen and for legacy context and internal 64b register is defined to keep the page table based cacheability. It is compromising eight (8) 8-bit entries (entry *i* comprises bits 8*i*+7:8*i* of the register).

For any access to a physical address, the table combines the memory type specified for that physical address by the MTRRs with a memory type selected from the PAT. Specifically, it comes from entry *i* of the PAT, where *i* is defined as follows:

- For an access to an entry in a paging structure whose address is in CR3 (e.g., the PML4 table with IA-32e paging):

i = 2*PCD+PWT, where the PCD and PWT values come from CR3.

- For an access to a paging-structure entry X whose address is in another paging structure entry Y, *i* = 2*PCD+PWT, where the PCD and PWT values come from Y.
- For an access to the physical address that is the translation of a linear address, *i* = 4*PAT+2*PCD+PWT, where the PAT, PCD, and PWT values come from the relevant PTE (if the translation uses a 4-KByte page), the relevant PDE (if the translation uses a 2-MByte page or a 4-MByte page), or the relevant PDPTE (if the translation uses a 1-GByte page).

PAT in Context Table Entry

PAT definition is embedded inside the context entry and already defined as part of the context entry definition (see related section). It allows 8 different settings which can be indexed using the following encodings listed as part of memory types. PAT in context entry is used for advanced context usage.

Memory Types and Applicability to GFX

The Memory Types defined for IA are listed below as:

Memory Type	Encoding in MTRR/PAT
Uncacheable (UC)	00h
Write Combining (WC)	01h
Write Through (WT)	04h
Write Protected (WP)	05h
WriteBack (WB)	06h
Uncached (UC-)	07h
Reserved*	02, 03, 08h-FFh

Note: * use of any reserved encodings will result in a FAULT and reported into fault registers.

- **Uncacheable (UC):** IA semantics for a UC cycle is slightly different than traditional UC concept that was adapted by GFX as part of integration into CPU. When UC type is selected from the MTRR table, GAM will enforce the request to be uncacheable in LLC/eLCC (turn-off the cacheability flags) and also force the fence semantics in GTI.

  Note: This behavior is not followed with a fence in case of GPUs.
- **Uncached (UC-)**: Same concept as UC from behavior perspective however the precedence can be overridden by WC unlike UC.
- **Write Combining (WC):** Write combining follows a streaming model in IA terms which is not cached in uncore. Semantically the existing GT use of UC concept overlaps with WC memory type defined by IA. GFX will treat the WC memory type as a streaming uncacheable memory type in GFX pipelines.
- **Write Through (WT):** Write through concept is already introduced as part of the gen7.5 design, the IA version of the WT overlaps with the same concept.
- **Write Protected (WP):** GFX has no concept of write protected; however this is simply a combination of two modes distributed over different access types:
  - **Reads**: Acts as WB.
  - **Writes**: Acts as WC.
- **Write Back (WB):** WB memory type is traditional memory type used where accesses are cached in uncore as per the directives provided. This is the main cacheable mode that will be used.

Basically, GPU will support all memory-types that CPU supports (as below), with the same meaning with respect to "**Cacheable**", "**Writeback Cacheable**", and "**Serialization**". Because the GPU has its own device-specific notion of what "*speculative processor ordering*" means, those are not specified/attached to memory-types.

Memory Type and Mnemonic	Cacheable	Writeback Cacheable	Allows Speculative Reads	Memory Ordering Model
Strong Uncacheable (UC)	No	No	No	Strong Ordering
Uncacheable (UC-)	No	No	No	Strong Ordering. Can only be selected through the PAT. Can be overridden by WC in MTRRs.
Write Combining (WC)	No	No	Yes	Weak Ordering. Available by programming MTRRs or by selecting it through the PAT.
Write Through (WT)	Yes	No	Yes	Speculative Processor Ordering.
Write Back (WB)	Yes	Yes	Yes	Speculative Processor Ordering.
Write Protected (WP)	Yes for reads; no for writes	No	Yes	Speculative Processor Ordering. Available by programming MTRRs.

The key similarities with CPU memory types are:

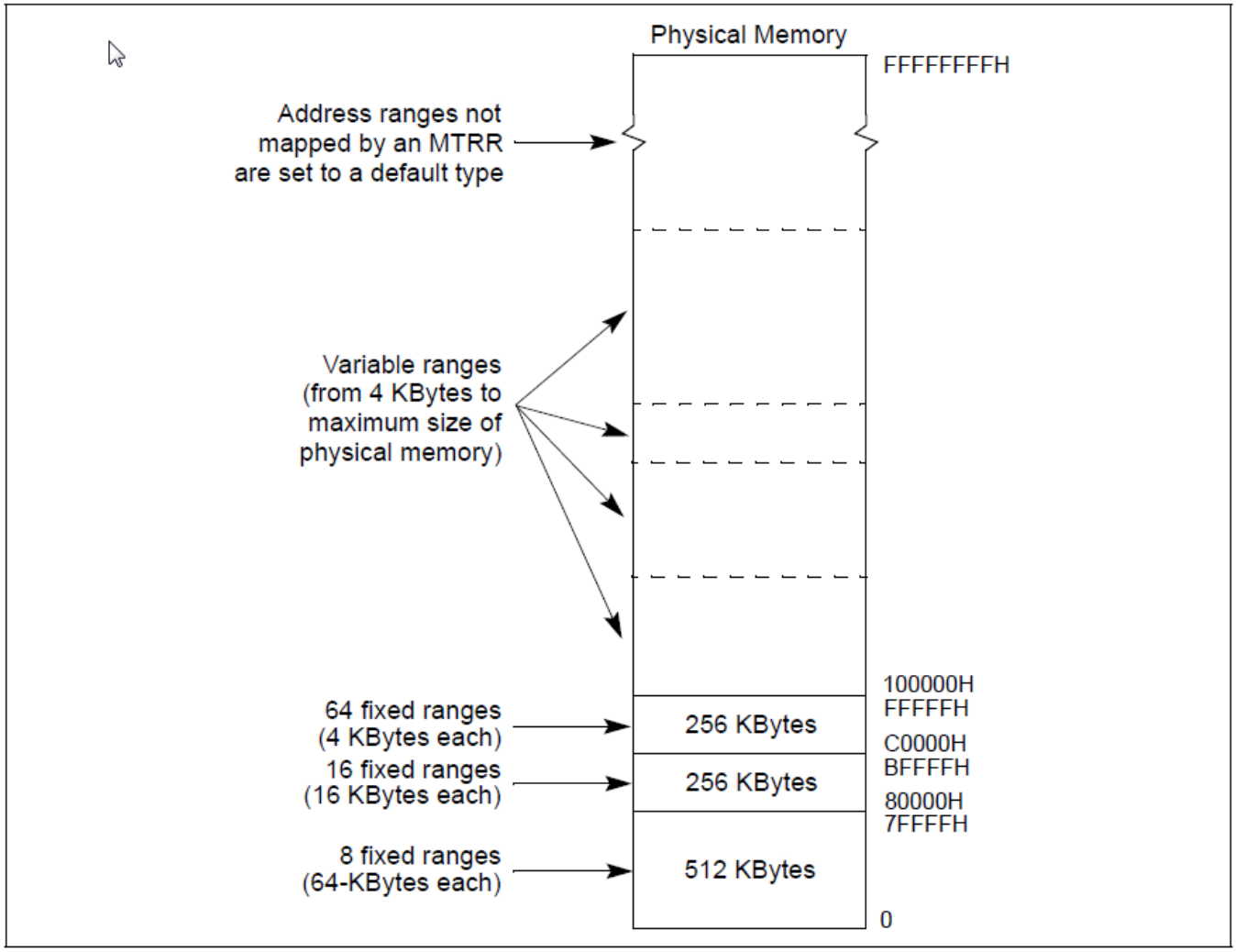
- GPU's cache (L3 cache) will be made coherent (works similar to MLC on CPUs).
- All GPU accesses (regardless of memory type) will snoop LLC.
- All GPU accesses will self-snoop.
- Outside of virtualization, GPU computes effective memory-type same as CPU (MTRR, PAT, PCD, PWT, etc.), and caches follow normal versus non-allocating mode per CD bit (similar to CPUs).
- UC, UC- and WC accesses do not allocate to caches. Will invalidate if line already exists in caches.
- UC- works same ways as CPU (i.e., unlike UC, UC- allows override by WC).
- WB accesses will allocate to cache without updating memory.
- WT behaves same as WB, except writes, updates memory along with allocate to cache.
- WP behaves same as WT, except writes always propagate to memory (invalidating any cache-line that hits).

The key difference with CPU memory types are:

- Speculative processor ordering not specified
- For GPU, only difference between UC/UC- and WC is that, UC has stronger ordering.

MTRR Ranges

Memory Type Range Registers are defined to cover the entire physical memory. The following table shows how each region is defined and how they map over the physical memory.



Memory Type Selection and Priority

Memory typing determination is split up to two categories:

- **Architectural Options**: This is the traditional memory typing defined via external specifications and controls the cacheability of various surfaces.
- **Design Specific Options**: To target product specific caches and cache optimizations

Design-Specific Memory Types

The following are the design specific memory types for CHV, BSW.

LRU Age: Both LLC and eDRAM uses LRU-like replacement algorithm with Age based determination

00: Age is 0

01: Age is 1

10: Age is 2

11: Age is 3

Target Cache: CHV, BSW has two large caches in Uncore where they could be separately targeted.

00: reserved

01: eLLC only

10: LLC only

11: eLLC/LLC

Both LRU and target cache selections can only be managed via non-architectural solutions.

Legacy Context: Selection is based on whether target cache field programmed in surface state is "00" or non-"00". If non-"00", than both the Age and Target cache parameters are picked from surface state and page table controls are ignored. If target cache parameter in surface state is "00" than, only page table controls are used via private PAT programming (see PAT calculation) and indexed into "PAT in MMIO Register Space".

Advanced Context: Decision is based on surface state only. Advanced context uses architectural definition of PAT via IA32e page tables which do not carry design specific information. The only mechanism to control cache LRU and targets is to program the surfaces state accordingly.

Memory Object Control State (Surface)

Bit	Description
6:5	**Memory Type: LLC/eLLC Cacheability Control (LeLLCCC)** This is the field used in GT interface block to determine what type of access need to be generated to uncore. For the cases where the LeLLCCC is set, cacheable transaction are generated to enable LLC usage for particular stream. 00: Use Cacheability Controls from page table / UC with Fence (if coherent cycle) 01/10: non-snooped 11: snooped **For CHV, BSW, GFX driver should use snooped type for only surfaces that are prepared by the driver in IA WB space. All other surfaces should be tagged as nonsnooped**
4:3	**Target Cache (TC)** This field allows the choice of LLC vs eLLC for caching

	00: eLLC Only – not snooped in GT 01: LLC Only 10: LLC/eLLC Allowed 11: L3, LLC, eLLC Allowed
2	**Reserved**
1:0	**Age for QUADLRU (AGE)** This field allows the selection of AGE parameter for a given surface in LLC or eLLC. . If a particular allocation is done at youngest age (“3”) it tends to stay longer in the cache as compared to older age allocations (“2”, “1”, or “0”). This option is given to driver to be able to decide which surfaces are more likely to generate HITs, hence need to be replaced least often in caches. 11: Good chance of generating hits. 10: Next good chance of generating hits 01: Decent chance of generating hits 00: Poor chance of generating hits

Architectural Memory Types

Memory typing is decided via several levels of checks and comparing different priority levels. The following table shows a visual mapping between these selections.

Context Type	2nd Level Translation	CD	EMTE	IGPT	Surface State	Private PAT	MTRR	IA32e PAT	EMT	Effective Memory Type
Legacy Context	X	X	X	X	UC	UC	X	X	X	UC
	X	X	X	X		WC	X	X	X	WC
	X	X	X	X		WT	X	X	X	WT
	X	X	X	X		WB	X	X	X	WB
	X	X	X	X	WC	UC	X	X	X	WC
	X	X	X	X		WC	X	X	X	
	X	X	X	X		WT	X	X	X	
	X	X	X	X		WB	X	X	X	
	X	X	X	X	WT	UC	X	X	X	WT
	X	X	X	X		WC	X	X	X	
	X	X	X	X		WT	X	X	X	
	X	X	X	X		WB	X	X	X	
	X	X	X	X	WB	UC	X	X	X	WB
	X	X	X	X		WC	X	X	X	
	X	X	X	X		WT	X	X	X	
	X	X	X	X		WB	X	X	X	
Advanced Context	0 (disabled)	0	X	X	X	X	UC	UC	X	UC
			X	X	X	X		UC-	X	UC
			X	X	X	X		WC	X	WC
			X	X	X	X		WT	X	UC
			X	X	X	X		WB	X	UC
			X	X	X	X		WP	X	UC
			X	X	X	X	WC	UC	X	UC
			X	X	X	X		UC-	X	WC
			X	X	X	X		WC	X	WC
			X	X	X	X		WT	X	UC
			X	X	X	X		WB	X	WC
			X	X	X	X		WP	X	UC
			X	X	X	X	WT	UC	X	UC
			X	X	X	X		UC-	X	UC
			X	X	X	X		WC	X	WC
			X	X	X	X		WT	X	WT
			X	X	X	X		WB	X	WT
			X	X	X	X		WP	X	WP
			X	X	X	X	WB	UC	X	UC
			X	X	X	X		UC-	X	UC
			X	X	X	X		WC	X	WC
			X	X	X	X		WT	X	WT
			X	X	X	X		WB	X	WB
			X	X	X	X		WP	X	WP
			X	X	X	X	WP	UC	X	UC
			X	X	X	X		UC-	X	WC
			X	X	X	X		WC	X	WC
			X	X	X	X		WT	X	WT
			X	X	X	X		WB	X	WP
			X	X	X	X		WP	X	WP

Context Type	2nd Level Translation	CD	EMTE	IGPT	Surface State	Private PAT	MTRR	IA32e PAT	EMT	Effective Memory Type
Advanced Context	1 (enabled)	0	0	X	X	X	UC	UC	X	UC
				X	X	X		UC-	X	UC
				X	X	X		WC	X	WC
				X	X	X		WT	X	UC
				X	X	X		WB	X	UC
				X	X	X		WP	X	UC
				X	X	X	WC	UC	X	UC
				X	X	X		UC-	X	WC
				X	X	X		WC	X	WC
				X	X	X		WT	X	UC
				X	X	X		WB	X	WC
				X	X	X		WP	X	UC
				X	X	X	WT	UC	X	UC
				X	X	X		UC-	X	UC
				X	X	X		WC	X	WC
				X	X	X		WT	X	WT
				X	X	X		WB	X	WT
				X	X	X		WP	X	WP
				X	X	X	WB	UC	X	UC
				X	X	X		UC-	X	UC
				X	X	X		WC	X	WC
				X	X	X		WT	X	WT
				X	X	X		WB	X	WB
				X	X	X		WP	X	WP
				X	X	X	WP	UC	X	UC
				X	X	X		UC-	X	WC
				X	X	X		WC	X	WC
				X	X	X		WT	X	WT
				X	X	X		WB	X	WP
				X	X	X		WP	X	WP
Advanced Context	1 (enabled)	0	1	0	X	X	X	UC	UC	UC
					X	X	X	UC-		UC
					X	X	X	WC		WC
					X	X	X	WT		UC
					X	X	X	WB		UC
					X	X	X	WP		UC
					X	X	X	UC	WC	UC
					X	X	X	UC-		UC
					X	X	X	WC		WC
					X	X	X	WT		UC
					X	X	X	WB		WC
					X	X	X	WP		UC
					X	X	X	UC	WT	UC
					X	X	X	UC-		UC
					X	X	X	WC		WC
					X	X	X	WT		WT
					X	X	X	WB		WT
					X	X	X	WP		WP
					X	X	X	UC	WB	UC
					X	X	X	UC-		UC
					X	X	X	WC		WC
					X	X	X	WT		WT
					X	X	X	WB		WB
					X	X	X	WP		WP
					X	X	X	UC	WP	UC
					X	X	X	UC-		WC
					X	X	X	WC		WC
					X	X	X	WT		WT
					X	X	X	WB		WP
					X	X	X	WP		WP
Advanced Context	1 (enabled)	0	1	1	X	X	X	X	UC	UC
					X	X	X	X	WC	WC
					X	X	X	X	WT	WT
					X	X	X	X	WB	WB
					X	X	X	X	WP	WP

Context Type	2nd Level Translation	CD	EMTE	IGPT	Surface State	Private PAT	MTRR	IA32e PAT	EMT	Effective Memory Type
Advanced Context	X	1	X	X	X	X	X	X	X	UC

Page Walker Access and Memory Types

Most of these notes are further explained in the document; however summarized as part of the page table behavior:

Page Walker Memory Types

1. Legacy Contexts
    a. GT access to root/extended-root table and context/extended-context table
    b. GTT access to private paging (PPGTT) entries
    c. GT access to GPA-to-HPA paging entries
    d. GT access to the translated page
2. Advanced context (without nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
3. Advanced context (with nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
    e. GT access to GPA-to-HPA paging entries to translate address of PASID-entry and PASID-state entry
    f. GT access to GPA-to-HPA paging entries to translate address of IA-32e paging entries
    g. GT access to GPA-to-HPA paging entries to translate address of page

For gen8, the following behavior is defined:

Error Cases

- A/D bit update attempt for paging entry in non-WB memory, cause page-walk to be aborted; Error reported to device in Translation Response, gets reported to driver as GPGPU context in error – catastrophic error case.
- Locked/Atomic operations to pages in non-WB memory aborted; gets reported to driver as GPGPU context in error (catastrophic error)
- CD=1 treated same as non-WB memory, for above lock behavior

Replacement

TLB replacements during runtime are based on LRA algorithm; in addition, invalidations and page responses will have to invalidate the TLB entries.

Invalidations of TLB

There are various ways to invalidate TLBs:

1. **Traditional invalidation from command streamer**: Could be part of any fence accesses including newly added atomics.
2. **SVM based invalidations**: Listed as part of the new SVM related invalidations, various stages of TLBs including intermediate stages can be invalidated selectively and/or as a whole.
3. **Context Switch**: A context switch has to invalidate caches to make sure we have no residual value of the TLBs across multiple PASIDs. GAM will treat the context reload message from CS as a form of TLB invalidation.
4. **A page response**: should invalidate faulted recordings. It should be done via address matching to kick the faulted entries within the matching PASID.

Invalidation response “Invalidation Wait Descriptor” should also be a fence for both READs and WRITEs that used the previous TLB entries. Gam can only respond to “invalidation wait descriptor” after getting a GTI EMPTY indication.

Optional Invalidations

The following cases are listed as page table updates which software may choose not to invalidate the TLBs.

- If a paging-structure is modified to change the Present (Valid) flag from 0 to 1, s/w may choose not to invalidate TLBs. This affects only the case where GPU keeps the faulted page in its TLB to filter out future faults. Regardless of s/w does invalidation or not, for the cases where h/w cares, there will be a page response from s/w which will be used to shootdown the faulted record from the TLB.

  GAM will only put faulted entries to its TLBs if there has been page request for it. This would mean only faultable surfaces can be stored in GAM TLBs as a faulted entry.

- If a paging-structure entry is modified to change the accessed flag from 0 to 1,no invalidation is necessary (assuming that an invalidation was performed the last time the accessed flag was changed from 1 to 0). This is because no TLB entry or paging-structure cache entry is created with information from a paging structure entry in which the accessed flag is 0.
- If a paging-structure entry is modified to change the R/W or U/S or XD flag from 0 to 1, failure to perform an invalidation may result in a "spurious" page-fault exception (e.g., in response to an attempted write access) but no other adverse behavior. Such an exception will occur at most once for each affected linear address

Faulting

Page Faults

CHV, BSW does not support page faulting

Memory Types and Cache Interface

This section has additional information on the types of memory which are accessible via the various GT mechanisms. It includes discussion on how the various paging models are used and accessed. See the Graphics Translation Tables for more detailed discussions on paging models.

This section also includes descriptions of how different surface types (MOCS) can be cached in the L3 and the different behaviors which can be enabled.

Memory Object Control State (MOCS)

The memory object control state defines the behavior of memory accesses beyond the graphics core, including encryption, graphics data types that allow selective flushing of data from outer caches, and controlling cacheability in the outer caches.

This control uses several mechanisms. Control state for all memory accesses can be defined page by page in the GTT entries. Memory objects that are defined by state per surface generally have additional memory object control state in the state structure that defines the other surface attributes. Memory objects without state defining them have memory object state control defined per class in the STATE_BASE_ADDRESS command, with class divisions the same as the base addresses. Finally, some memory objects only have the GTT entry mechanism for defining this control. The table below enumerates the memory objects and the location of the control state for each:

Memory Object	Location of Control State
surfaces defined by SURFACE_STATE: sampling engine surfaces, render targets, media surfaces, pull constant buffers, streamed vertex buffers	SURFACE_STATE
depth, stencil, and hierarchical depth buffers	corresponding state command that defined the buffer attributes
stateless buffers accessed by data port	STATE_BASE_ADDRESS
indirect state objects	STATE_BASE_ADDRESS
kernel instructions	STATE_BASE_ADDRESS
push constant buffers	3DSTATE_CONSTANT_(VS | GS | PS)
index buffers	3DSTATE_INDEX_BUFFER
vertex buffers	3DSTATE_VERTEX_BUFFERS
indirect media object	STATE_BASE_ADDRESS
generic state prefetch	GTT control only
ring/batch buffers	GTT control only
context save buffers	GTT control only
store DWord	GTT control only

MOCS Registers

These registers provide the detailed format of the MOCS table entries that need to be programmed to define each surface state.

MEMORY_OBJECT_CONTROL_STATE

MEMORY_OBJECT_CONTROL_STATE

Page Walker Access and Memory Types

Most of these notes are further explained in the document however summarized as part of the page table behavior:

Page Walker Memory Types

1. Legacy Contexts
    a. GT access to root/extended-root table and context/extended-context table
    b. GTT access to private paging (PPGTT) entries
    c. GT access to GPA-to-HPA paging entries
    d. GT access to the translated page
2. Advanced context (without nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
3. Advanced context (with nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
    e. GT access to GPA-to-HPA paging entries to translate address of PASID-entry and PASID-state entry
    f. GT access to GPA-to-HPA paging entries to translate address of IA-32e paging entries
    g. GT access to GPA-to-HPA paging entries to translate address of page

Gen8 Memory Typing for Paging

The following information is duplicated in the Page Walker Memory Types topic:

1. Legacy Contexts
    a. GT access to root/extended-root table and context/extended-context table
    b. GTT access to private paging (PPGTT) entries
    c. GT access to GPA-to-HPA paging entries
    d. GT access to the translated page
2. Advanced context (without nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
3. Advanced context (with nesting)
    a. GT access to extended-root table and extended-context table
    b. GT access to PASID-entry & PASID-state entry
    c. GT access to IA-32e paging entries
    d. GT access to the translated page
    e. GT access to GPA-to-HPA paging entries to translate address of PASID-entry and PASID-state entry
    f. GT access to GPA-to-HPA paging entries to translate address of IA-32e paging entries
    g. GT access to GPA-to-HPA paging entries to translate address of page

This information is new in this topic and references the cases and subcases enumerated above:

For case [1]:

- [1.a] is always covered as a non-cacheable access
- [1.b] & [1.c] is covered with MMIO register where PPGTT entries can be forced to be cached in LLC (default option is cached).
- [1.d] is defined via private PAT (MMIO based) and surface state.

For case [2]:

- [2.a] is always covered as a non-cacheable access
- [2.b] is always cached & PASID state table entry is always accessed “atomically”
- [2.c] is accessed as cached
- [2.d] use memory-type as evaluated through MTRR, CD, and PCD/PWT/PAT bits in leaf IA-32e paging entry

For case [3]:

- [3.a] is always covered as a non-cacheable access
- [3.b] is always cached & PASID state table entry is always accessed “atomically”
- [3.c] is accessed as cached
- [3.d] use memory-type as follows (this section is further described in detail in memory typing section)
    - If CD=1, memory-type is UC
    - If CD=0:
        - If EMTE=0 in extended-context-entry, it is handled same as [2.d]
        - If EMTE=1 in extended-context-entry:
            - If IGMT=1 in leaf GPA-to-HPA entry, memory type used is the EMT field in this GPA-to-HPA entry.
            - If IGMT=0 in leaf GPA-to-HPA entry, memory type from [2.d] is combined with EMT field in this GPA-to-HPA entry.
- [3.e] is always cached & PASID state table entry is always accessed “atomically”
- [3.f] &[3.g] is accessed as cached

Error Cases

- A/D bit update attempt for paging entry in non-WB memory, causes page-walk to be aborted; Error reported to device in Translation Response; For Gen, gets reported to driver as GPGPU context in error – catastrophic error case.
- Locked/Atomic operations to pages in non-WB memory aborted; For Gen, gets reported to driver as GPGPU context in error (catastrophic error).
- CD=1 treated same as non-WB memory, for above lock behavior.

Common Surface Formats

This section documents surfaces and how they are stored in memory, including 3D and video surfaces, including the details of compressed texture formats. Also covered are the surface layouts based on tiling mode and surface type.

Non-Video Surface Formats

This section describes the lowest-level organization of a surfaces containing discrete “pixel” oriented data (e.g., discrete pixel (RGB,YUV) colors, subsampled video data, 3D depth/stencil buffer pixel formats, bump map values etc. Many of these pixel formats are common to the various pixel-oriented memory object types.

Surface Format Naming

Unless indicated otherwise, all pixels are **stored** in “**little endian**” byte order. i.e., pixel bits 7:0 are stored in byte *n*, pixel bits 15:8 are stored in byte *n*+1, and so on. The format labels include color components in little endian order (e.g., R8G8B8A8 format is physically stored as R, G, B, A).

The name of most of the surface formats specifies its format. Channels are listed in little endian order (LSB channel on the left, MSB channel on the right), with the channel format specified following the channels with that format. For example, R5G5_SNORM_B6_UNORM contains, from LSB to MSB, 5 bits of red in SNORM format, 5 bits of green in SNORM format, and 6 bits of blue in UNORM format.

Intensity Formats

All surface formats containing “I” include an intensity value. When used as a source surface for the sampling engine, the intensity value is replicated to all four channels (R,G,B,A) before being filtered. Intensity surfaces are not supported as destinations.

Luminance Formats

All surface formats containing “L” include a luminance value. When used as a source surface for the sampling engine, the luminance value is replicated to the three color channels (R,G,B) before being filtered. The alpha channel is provided either from another field or receives a default value. Luminance surfaces are not supported as destinations.

R1_UNORM (same as R1_UINT) and MONO8

When used as a texel format, the R1_UNORM format contains 8 1-bit Intensity (I) values that are replicated to all color channels. Note that T0 of byte 0 of a R1_UNORM-formatted texture corresponds to Texel[0,0]. This is different from the format used for monochrome sources in the BLT engine.

7	6	5	4	3	2	1	0
T7	T6	T5	T4	T3	T2	T1	T0

Bit	Description
T0	**Texel 0** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1
...	**...**
T7	**Texel 7** On texture reads, this (unsigned) 1-bit value is replicated to all color channels. Format: U1

MONO8 format is identical to R1_UNORM but has different semantics for filtering. MONO8 is the only supported format for the MAPFILTER_MONO filter. See the *Sampling Engine* chapter.

Palette Formats

Palette formats are supported by the sampling engine. These formats include an index into the palette (Px) that selects the actual channel values from the palette, which is loaded via the 3DSTATE_SAMPLER_PALETTE_LOAD0 command.

P4A4_UNORM

This surface format contains a 4-bit Alpha value (in the high nibble) and a 4-bit Palette Index value (in the low nibble).

7			4	3			0
Alpha				Palette Index			

Bit	Description
7:4	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] Alpha value. Format: U4
3:0	**Palette Index** A 4-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U4

A4P4_UNORM

This surface format contains a 4-bit Alpha value (in the low nibble) and a 4-bit Color Index value (in the high nibble).

7			4	3			0
Palette Index				Alpha			

Bit	Description
7:4	**Palette Index** A 4-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U4
3:0	**Alpha** Alpha value which will be replicated to both the high and low nibble of an 8-bit value, and then divided by 255 to yield a [0.0,1.0] alpha value.

Bit	Description
	Format: U4

P8A8_UNORM

This surface format contains an 8-bit Alpha value (in the high byte) and an 8-bit Palette Index value (in the low byte).

15			8	7			0
Alpha				Palette Index			

Bit	Description
15:8	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] Alpha value. Format: U8
7:0	**Palette Index** An 8-bit index which is used to lookup a 24-bit (RGB) value in the texture palette (loaded via 3DSTATE_SAMPLER_PALETTE_LOADx) Format: U8

A8P8_UNORM

This surface format contains an 8-bit Alpha value (in the low byte) and an 8-bit Color Index value (in the high byte).

15			8	7			0
Palette Index				Alpha			

Bit	Description
15:8	**Palette Index** An 8-bit color index which is used to lookup a 24-bit RGB value in the texture palette. Format: U8
7:0	**Alpha** Alpha value which will be divided by 255 to yield a [0.0,1.0] alpha value. Format: U8

P8_UNORM

This surface format contains only an 8-bit Color Index value.

Bit	Description
7:0	**Palette Index** An 8-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U8

P2_UNORM

This surface format contains only a 2-bit Color Index value.

Bit	Description
1:0	**Palette Index** A 2-bit color index which is used to lookup a 32-bit ARGB value in the texture palette. Format: U2

Compressed Surface Formats

This section contains information on the internal organization of compressed surface formats.

ETC1_RGB8

CHV, BSW: This format compresses UNORM RGB data using an 8-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows.

High 24 bits if "diff" is zero (individual mode):

Bits	Description
7:4	R0[3:0]
3:0	R1[3:0]
15:12	G0[3:0]
11:8	G1[3:0]
23:20	B0[3:0]
19:16	B1[3:0]

High 24 bits if "diff" is one (differential mode):

Bits	Description
7:3	R0[4:0]
2:0	dR1[2:0]
15:11	G0[4:0]
10:8	dG1[2:0]
23:19	B0[4:0]
18:16	dB1[2:0]

Low 40 bits:

Bits	Description
31:29	lum table index for sub-block 0
28:26	lum table index for sub-block 1
25	diff
24	flip
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB
36	texel[0][3] index MSB
35	texel[3][2] index MSB

Bits	Description
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB
40	texel[0][0] index MSB
55	texel[3][3] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The 4x4 is divided into two 8-pixel sub-blocks, either two 2x4 sub-blocks or two 4x2 sub-blocks controlled by the “flip” bit. If flip=0, sub-block 0 is the 2x4 on the left and sub-block 1 is the 2x4 on the right. If flip=1, sub-block 0 is the 4x2 on the top and sub-block 1 is the 4x2 on the bottom.

The “diff” bit controls whether the red/green/blue values (R0/G0/B0/R1/G1/B1) are stored as one 444 value per sub-block (“individual” mode with diff = 0), or a single 555 value for the first sub-block (R0/G0/B0) and a 333 delta value (dR1/dG1/dB1) for the second sub-block (“differential” mode with diff = 1). The delta values are 3-bit two’s-complement values that hold values in the range [-4,3]. These values are added to the 5-bit values for sub-block 0 to obtain the 5-bit values for sub-block 1 (if the value is outside of the range [0,31], the result of the decompression is undefined). From the 4- or 5-bit

per channel values, an 8-bit value for each channel is extended by replication and provides the 888 base color for each sub-block.

For each sub-block one of 8 different luminance columns is selected based on the 3-bit lum table index. Then each texel selects one of the 4 rows of the selected column with a 2-bit per-texel index. The chosen value in the table is added to the 8-bit base color for the sub-block (obtained in the previous step) to obtain the texel's color. Values in the table are given in decimal, representing an 8-bit UNORM as an 8-bit signed integer.

Luminance Table

	0	1	2	3	4	5	6	7
0	2	5	9	13	18	24	33	47
1	8	17	29	42	60	80	106	183
2	-2	-5	-9	-13	-18	-24	-33	-47
3	-8	-17	-29	-42	-60	-80	-106	-183

ETC2_RGB8 and ETC2_SRGB8

The ETC2_RGB8 format builds on top of ETC1_RGB8, using a set of invalid bit sequences to enable three new modes. The two modes of ETC1_RGB8 are also supported with ETC2_RGB8, and will not be documented in this section as they are covered in the ETC1_RGB8 section.

The detection of the three new modes is based on RGB and diff bits in locations as defined for ETC1 differential mode. The mode is determined as follows (x indicates don't care):

diff	Rt	Gt	Bt	mode
0	x	x	x	individual
1	0	x	x	T
1	1	0	x	H
1	1	1	0	planar
1	1	1	1	differential

The inputs in the above table are defined as follows:

```
Rt = (R0 + dR1) in [0,31]
Gt = (G0 + dG1) in [0,31]
Bt = (G0 + dB1) in [0,31]
```

8-byte compression block for mode determination

Bits	Description
7:3	R0[4:0]
2:0	dR1[2:0]
15:11	G0[4:0]
10:8	dG1[2:0]
23:19	B0[4:0]
18:16	dB1[2:0]
31:26	ignored
25	diff
24	ignored
63:32	ignored

The fields in the table above are used *only* for mode determination. Some of the bits in this table are overloaded with other values within each mode. The algorithm is defined such that there is no ambiguity in modes when this is done.

T mode

The "T" mode has the following bit definition:

8-byte compression block for "T" mode

Bits	Description
7:5	ignored
4:3	R0[3:2]
2	ignored
1:0	R0[1:0]
15:12	G0[3:0]
11:8	B0[3:0]
23:20	R1[3:0]
19:16	G1[3:0]
31:28	B1[3:0]
27:26	di[2:1]
25	diff = 1
24	di[0]
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB
36	texel[0][3] index MSB
35	texel[3][2] index MSB
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB
40	texel[0][0] index MSB
55	texel[0][0] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB

Bits	Description
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The “T” mode has two base colors stored as 4 bits per channel, R0/G0/B0 and R1/G1/B1, as in the individual mode, however the bit positions for these are different. For each channel, the 4 bits are extended to 8 bits by bit replication.

A 3-bit distance index “di” is also defined in the compression block. This value is used to look up the distance in the following table:

distance index “di”	distance “d”
0	3
1	6
2	11
3	16
4	23
5	32
6	41
7	64

Four colors are possible on each texel. These colors are defined as the following:

```
P0 = (R0, G0, B0)
P1 = (R1, G1, B1) + (d, d, d)
P2 = (R1, G1, B1)
P3 = (R1, G1, B1) – (d, d, d)
```

All resulting channels are clamped to the range [0,255]. One of the four colors is then assigned to each texel in the block based on the 2-bit texel index.

H mode

The “H” mode has the following bit definition:

8-byte compression block for “H” mode

Bits	Description
7	ignored
6:3	R0[3:0]
2:0	G0[3:1]
15:13	ignored
12	G0[0]
11	B0[3]
10	ignored
9:8	B0[2:1]
23	B0[0]
22:19	R1[3:0]
18:16	G1[3:1]
31	G1[0]
30:27	B1[3:0]
26	di[2]
25	diff = 1
24	di[1]
39	texel[3][3] index MSB
38	texel[2][3] index MSB
37	texel[1][3] index MSB
36	texel[0][3] index MSB
35	texel[3][2] index MSB
34	texel[2][2] index MSB
33	texel[1][2] index MSB
32	texel[0][2] index MSB
47	texel[3][1] index MSB
46	texel[2][1] index MSB
45	texel[1][1] index MSB
44	texel[0][1] index MSB
43	texel[3][0] index MSB
42	texel[2][0] index MSB
41	texel[1][0] index MSB

Bits	Description
40	texel[0][0] index MSB
55	texel[3][3] index LSB
54	texel[2][3] index LSB
53	texel[1][3] index LSB
52	texel[0][3] index LSB
51	texel[3][2] index LSB
50	texel[2][2] index LSB
49	texel[1][2] index LSB
48	texel[0][2] index LSB
63	texel[3][1] index LSB
62	texel[2][1] index LSB
61	texel[1][1] index LSB
60	texel[0][1] index LSB
59	texel[3][0] index LSB
58	texel[2][0] index LSB
57	texel[1][0] index LSB
56	texel[0][0] index LSB

The “H” mode has two base colors stored as 4 bits per channel, R0/G0/B0 and R1/G1/B1, as in the individual and T modes, however the bit positions for these are different. For each channel, the 4 bits are extended to 8 bits by bit replication.

A 3-bit distance index “di” is defined by 2 MSBs in the compression block and the LSB computed by the following equation, where R/G/B values are the 8-bit values from the first step:

```
di[0] = ((R0 « 16) | (G0 « 8) | B0) >= ((R1 « 16) | (G1 « 8) | B1)
```

The distance “d” is then looked up in the same table used for T mode. The four colors for H mode are computed as follows:

```
P0 = (R0, G0, B0) + (d, d, d)
P1 = (R0, G0, B0) - (d, d, d)
P2 = (R1, G1, B1) + (d, d, d)
P3 = (R1, G1, B1) - (d, d, d)
```

All resulting channels are clamped to the range [0,255]. One of the four colors is then assigned to each texel in the block based on the 2-bit texel index as in T mode.

Planar mode

The “planar” mode has the following bit definition:

8-byte compression block for “planar” mode

Bits	Description
7	ignored
6:1	R0[5:0]
0	G0[6]
15	ignored
14:9	G0[5:0]
8	B[5]
23:21	ignored
20:19	B[4:3]
18	ignored
17:16	B0[2:1]
31	B0[0]
30:26	RH[5:1]
25	diff = 1
24	RH[0]
39:33	GH[6:0]
32	BH[5]
47:43	BH[4:0]
42:40	RV[5:3]
55:53	RV[2:0]
52:48	GV[6:2]
63:62	GV[1:0]
61:56	BV[5:0]

The “planar” mode has three base colors stored as RGB 676, with red & blue having 6 bits and green having 7 bits. These three base colors are each extended to RGB 888 with bit replication.

The color of each texel is then computed using the following equations, with x and y representing the texel position within the compression block:

```
texel[y][x].R = x(RH-R0)/4 + y(RV-R0)/4 + R0
texel[y][x].G = x(GH-G0)/4 + y(GV-G0)/4 + G0
texel[y][x].B = x(BH-B0)/4 + y(BV-B0)/4 + B0
```

All resulting channels are clamped to the range [0,255].

The ETC2_SRGB8 format is decompressed as if it is ETC2_RGB8, then a conversion from the resulting RGB values to SRGB space is performed.

EAC_R11 and EAC_SIGNED_R11

These formats compress UNORM/SNORM single-channel data using an 8-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows.

EAC_R11 compression block layout

Bits	Description
7:0	R0[7:0]
15:12	m[3:0]
11:8	ti[3:0]
23:21	texel[0][0] index
20:18	texel[1][0] index
17:16,31	texel[2][0] index
30:28	texel[3][0] index
27:25	texel[0][1] index
24,39:38	texel[1][1] index
37:35	texel[2][1] index
34:32	texel[3][1] index
47:45	texel[0][2] index
44:42	texel[1][2] index
41:40,55	texel[2][2] index
54:52	texel[3][2] index
51:49	texel[0][3] index
48,63:62	texel[1][3] index
61:59	texel[2][3] index
58:56	texel[3][3] index

The “ti” (table index) value from the compression block is used to select one of the columns in the table below.

Intensity modifier (im) table

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	-3	-3	-2	-2	-3	-3	-4	-3	-2	-2	-2	-2	-3	-1	-4	-3
1	-6	-7	-5	-4	-6	-7	-7	-5	-6	-5	-4	-5	-4	-2	-6	-5
2	-9	-10	-8	-6	-8	-9	-8	-8	-8	-8	-8	-7	-7	-3	-8	-7
3	-15	-13	-13	-13	-12	-11	-11	-11	-10	-10	-10	-10	-10	-10	-9	-9
4	2	2	1	1	2	2	3	2	1	1	1	1	2	0	3	2
5	5	6	4	3	5	6	6	4	5	4	3	4	3	1	5	4
6	8	9	7	5	7	8	7	7	7	7	7	6	6	2	7	6
7	14	12	12	12	11	10	10	10	9	9	9	9	9	9	8	8

The eight possible color values R_i are then computed from the 8 values in the column labeled im_i, where i ranges from 0 to 7:

For EAC_R11:

```
if (m == 0) Ri = R0*8 + 4 + imi else Ri = R0*8 + 4 + (imi * m * 8)
```

Each value is clamped to the range [0,2047].

For EAC_SIGNED_R11:

```
if (m == 0) Ri = R0*8 + imi else Ri = R0*8 + (imi * m * 8)
```

Each value is clamped to the range [-1023,1023].

Note that in the signed case, the R0 value is a signed, 2’s complement value in the range [-127, 127]. Before being used in the above equations, an R0 value of -128 must be clamped to -127.

Finally, each texel red value is selected from the 8 possible values R_i using the 3-bit index for that texel. The green, blue, and alpha values are set to their default values.

The final value represents an 11-bit UNORM or SNORM as an unsigned/signed integer.

ETC2_RGB8_PTA and ETC2_SRGB8_PTA

The ETC2_RGB8_PTA format is similar to ETC2_RGB8 but eliminates the “individual” mode in favor of allowing a punch-through alpha. The “diff” bit from ETC2_RGB8 is renamed to “opaque” in this format, and the mode selection behaves as if the “diff” bit is always 1, making the “individual” mode inaccessible for these formats.

An alpha value of either 0 or 255 (representing 0.0 or 1.0) is possible with this format. If alpha is determined to be zero, the three other channels are also forced to zero, regardless of what value the normal decompression algorithm would have produced.

Differential Mode

In differential mode, if the opaque bit is set, the luminance table for ETC2_RGB8 is used. If the opaque bit is not set, the following luminance table is used (note that rows 0 and 2 have been zeroed out, otherwise the table is the same):

Luminance Table for opaque bit not set

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	8	17	29	42	60	80	106	183
2	0	0	0	0	0	0	0	0
3	-8	-17	-29	-42	-60	-80	-106	-183

For each texel, if the opaque bit is zero and the corresponding texel index is equal to 2, the alpha value is set to zero (and therefore RGB for that texel will also end up at zero). Otherwise alpha is set to 255 and RGB is the result of the normal decompression calculations.

T and H Modes

In both of these modes, if the opaque bit is zero and the texel index is equal to 2, the alpha value is set to zero (and therefore RGB will also end up at zero). Otherwise alpha is set to 255.

Planar Mode

In planar mode, the opaque bit is ignored and alpha is set to 255.

The ETC2_SRGB8_PTA format is decompressed as if it is ETC2_RGB8_PTA, then a conversion from the resulting RGB values to SRGB space is performed, with alpha remaining unchanged.

ETC2_EAC_RGBA8 and ETC2_EAC_SRGB8_A8

The ETC2_EAC_RGBA8 format is a combination of ETC2_RGB8 and EAC_R8. A 16-byte compression block represents each 4x4. The low-order 8 bytes are used to compute alpha (instead of red) using the EAC_R8 algorithm. The high-order 8 bytes are used to compute RGB using the ETC2_RGB8 algorithm. The EAC_R8 format differs from EAC_R11 as described below.

The ETC2_EAC_SRGB8_A8 format is decompressed as if it is ETC2_EAC_RGBA8, then a conversion from the resulting RGB values to SRGB space is performed, with alpha remaining unchanged.

EAC_R8 Format:

The EAC_R8 format used within these surface formats is identical to EAC_R11 described in an earlier section, except the procedure for computing the eight possible color values Ri is performed as follows:

Ri = R0 + (imi * m)

Each value is clamped to the range [0,255].

EAC_RG11 and EAC_SIGNED_RG11

These formats compress UNORM/SNORM double-channel data using a 16-byte compression block representing a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 16-byte compression block is laid out as follows.

EAC_RG11 compression block layout

Bits	Description
63:56	G0[7:0]
55:52	Gm[3:0]
51:48	Gti[3:0]
47:45	texel[0][0] G index
44:42	texel[1][0] G index
41:39	texel[2][0] G index
38:36	texel[3][0] G index
35:33	texel[0][1] G index
32:30	texel[1][1] G index
29:27	texel[2][1] G index
26:24	texel[3][1] G index
23:21	texel[0][2] G index
20:18	texel[1][2] G index
17:15	texel[2][2] G index
14:12	texel[3][2] G index
11:9	texel[0][3] G index
8:6	texel[1][3] G index
5:3	texel[2][3] G index
66:64	texel[3][3] G index
63:56	R0[7:0]
55:52	Rm[3:0]
51:48	Rti[3:0]
47:45	texel[0][0] R index
44:42	texel[1][0] R index
41:39	texel[2][0] R index
38:36	texel[3][0] R index
35:33	texel[0][1] R index
32:30	texel[1][1] R index
29:27	texel[2][1] R index

Bits	Description
26:24	texel[3][1] R index
23:21	texel[0][2] R index
20:18	texel[1][2] R index
17:15	texel[2][2] R index
14:12	texel[3][2] R index
11:9	texel[0][3] R index
8:6	texel[1][3] R index
5:3	texel[2][3] R index
2:0	texel[3][3] R index

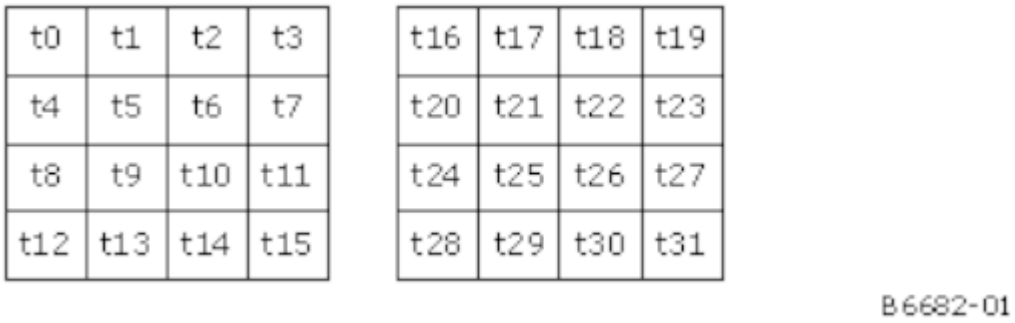
These compression formats are identical to the EAC_R11 and EAC_SIGNED_R11 formats, except that they supply two channels of output data, both red and green, from two independent 8-byte portions of the compression block. The low half of the compression block contains the red information, and the high half contains the green information. Blue and alpha channels are set to their default values.

Refer to the EAC_R11 and EAC_SIGNED_R11 specification for details on how the red and green channels are generated using the data in the compression block.

FXT Texture Formats

There are four different FXT1 compressed texture formats. Each of the formats compress two 4x4 texel blocks into 128 bits. In each compression format, the 32 texels in the two 4x4 blocks are arranged according to the following diagram:

FXT1 Encoded Blocks



Overview of FXT1 Formats

During the compression phase, the encoder selects one of the four formats for each block based on which encoding scheme results in best overall visual quality. The following table lists the four different modes and their encodings:

FXT1 Format Summary

Bit 127	Bit 126	Bit 125	Block Compression Mode	Summary Description
0	0	X	**CC_HI**	2 R5G5B5 colors supplied. Single LUT with 7 interpolated color values and transparent black
0	1	0	**CC_CHROMA**	4 R5G5B5 colors used directly as 4-entry LUT.
0	1	1	**CC_ALPHA**	3 A5R5G5B5 colors supplied. LERP bit selects between 1 LUT with 3 discrete colors + transparent black and 2 LUTs using interpolated values of Color 0,1 (t0-15) and Color 1,2 (t16-31).
1	x	x	**CC_MIXED**	4 R5G5B5 colors supplied, where Color0,1 LUT is used for t0-t15, and Color2,3 LUT used for t16-31. Alpha bit selects between LUTs with 4 interpolated colors or 3 interpolated colors + transparent black.

FXT1 CC_HI Format

In the CC_HI encoding format, two base 15-bit R5G5B5 colors (Color 0, Color 1) are included in the encoded block. These base colors are then expanded (using high-order bit replication) to 24-bit RGB colors, and used to define an 8-entry lookup table of interpolated color values (the 8th entry is transparent black). The encoded block contains a 3-bit index value per texel that is used to lookup a color from the table.

CC_HI Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_HI block format:

FXT CC_HI Block Encoding

Bit	Description
127:126	Mode = '00'b (CC_HI)
125:121	Color 1 Red
120:116	Color 1 Green
115:111	Color 1 Blue
110:106	Color 0 Red
105:101	Color 0 Green
100:96	Color 0 Blue
95:93	Texel 31 Select
...	...
50:48	Texel 16 Select
47:45	Texel 15 Select
...	...
2:0	Texel 0 Select

CC_HI Block Decoding

The two base colors, Color 0 and Color 1 are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

FXT CC_HI Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 1 [23:19]	Color 1 Red [7:3]	[125:121]
Color 1 [18:16]	Color 1 Red [2:0]	[125:123]
Color 1 [15:11]	Color 1 Green [7:3]	[120:116]
Color 1 [10:08]	Color 1 Green [2:0]	[120:118]
Color 1 [07:03]	Color 1 Blue [7:3]	[115:111]
Color 1 [02:00]	Color 1 Blue [2:0]	[115:113]
Color 0 [23:19]	Color 0 Red [7:3]	[110:106]
Color 0 [18:16]	Color 0 Red [2:0]	[110:108]
Color 0 [15:11]	Color 0 Green [7:3]	[105:101]
Color 0 [10:08]	Color 0 Green [2:0]	[105:103]
Color 0 [07:03]	Color 0 Blue [7:3]	[100:96]
Color 0 [02:00]	Color 0 Blue [2:0]	[100:98]

These two 24-bit colors (Color 0, Color 1) are then used to create a table of seven interpolated colors (with Alpha = 0FFh), along with an eight entry equal to RGBA = 0,0,0,0, as shown in the following table:

FXT CC_HI Interpolated Color Table

Interpolated Color	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(5 * Color0.RGB + 1 * Color1.RGB + 3) / 6	0FFh
2	(4 * Color0.RGB + 2 * Color1.RGB + 3) / 6	0FFh
3	(3 * Color0.RGB + 3 * Color1.RGB + 3) / 6	0FFh
4	(2 * Color0.RGB + 4 * Color1.RGB + 3) / 6	0FFh
5	(1 * Color0.RGB + 5 * Color1.RGB + 3) / 6	0FFh
6	Color1.RGB	0FFh
7	RGB = 0,0,0	0

This table is then used as an 8-entry Lookup Table, where each 3-bit Texel n Select field of the encoded CC_HI block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_HI block.

FXT1 CC_CHROMA Format

In the CC_CHROMA encoding format, four 15-bit R5B5G5 colors are included in the encoded block. These colors are then expanded (using high-order bit replication) to form a 4-entry table of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

CC_CHROMA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_CHROMA block format:

FXT CC_CHROMA Block Encoding

Bit	Description
127:125	Mode = '010'b (CC_CHROMA)
124	Unused
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	
33:32	Texel 16 Select
31:30	Texel 15 Select
...	
1:0	Texel 0 Select

CC_CHROMA Block Decoding

The four colors (Color 0-3) are converted from R5G5B5 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

FXT CC_CHROMA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10:08]	Color 3 Green [2:0]	[118:116]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

This table is then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_CHROMA block is used to index into a 32-bit A8R8G8B8 color from the table (Alpha defaults to 0FFh) completing the decode of the CC_CHROMA block.

FXT CC_CHROMA Interpolated Color Table

Texel Select	Color ARGB
0	Color0.ARGB
1	Color1.ARGB
2	Color2.ARGB
3	Color3.ARGB

FXT1 CC_MIXED Format

In the CC_MIXED encoding format, four 15-bit R5G5B5 colors are included in the encoded block: Color 0 and Color 1 are used for Texels 0-15, and Color 2 and Color 3 are used for Texels 16-31.

Each pair of colors are then expanded (using high-order bit replication) to form 4-entry tables of 24-bit RGB colors. The encoded block contains a 2-bit index value per texel that is used to lookup a 24-bit RGB color from the table. The Alpha component defaults to fully opaque (0FFh).

CC_MIXED Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_MIXED block format:

FXT CC_MIXED Block Encoding

Bit	Description
127	Mode = ‘1’b (CC_MIXED)
126	Color 3 Green [0]
125	Color 1 Green [0]
124	Alpha [0]
123:119	Color 3 Red
118:114	Color 3 Green
113:109	Color 3 Blue
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

CC_MIXED Block Decoding

The decode of the CC_MIXED block is modified by Bit 124 (Alpha [0]) of the encoded block.

Alpha[0] = 0 Decoding

When Alpha[0] = 0 the four colors are encoded as 16-bit R5G6B5 values, with the Green LSB defined as per the following table:

FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 2 Green [0]	Encoded Bit [33] XOR Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]
Color 0 Green [0]	Encoded Bit [1] XOR Encoded Bit [125]

The four colors (Color 0-3) are then converted from R5G5B6 to R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following table:

FXT CC_MIXED Decoded Colors (Alpha[0] = 0)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10]	Color 2 Green [2]	[33] XOR [126]]
Color 2 [09:08]	Color 2 Green [1:0]	[103:100]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10]	Color 0 Green [2]	[1] XOR [125]
Color 0 [09:08]	Color 0 Green [1:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four interpolated colors (with Alpha = 0FFh). The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices, as shown in the following figures:

FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(2*Color0.RGB + Color1.RGB + 1) /3	0FFh
2	(Color0.RGB + 2*Color1.RGB + 1) /3	0FFh
3	Color1.RGB	0FFh

FXT CC_MIXED Interpolated Color Table (Alpha[0]=0, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(2/3) * Color2.RGB + (1/3) * Color3.RGB	0FFh
2	(1/3) * Color2.RGB + (2/3) * Color3.RGB	0FFh
3	Color3.RGB	0FFh

Alpha[0] = 1 Decoding

When Alpha[0] = 1, Color0 and Color2 are encoded as 15-bit R5G5B5 values. Color1 and Color3 are encoded as RGB565 colors, with the Green LSB obtained as shown in the following table:

FXT CC_MIXED (Alpha[0]=0) Decoded Colors

Encoded Color Bit	Definition
Color 3 Green [0]	Encoded Bit [126]
Color 1 Green [0]	Encoded Bit [125]

All four colors are then expanded to 24-bit R8G8B8 colors by bit replication, as show in the following diagram.

FXT CC_MIXED Decoded Colors (Alpha[0] = 1)

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 3 [23:17]	Color 3 Red [7:3]	[123:119]
Color 3 [18:16]	Color 3 Red [2:0]	[123:121]
Color 3 [15:11]	Color 3 Green [7:3]	[118:114]
Color 3 [10]	Color 3 Green [2]	[126]
Color 3 [09:08]	Color 3 Green [1:0]	[118:117]
Color 3 [07:03]	Color 3 Blue [7:3]	[113:109]
Color 3 [02:00]	Color 3 Blue [2:0]	[113:111]
Color 2 [23:19]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10]	Color 1 Green [2]	[125]
Color 1 [09:08]	Color 1 Green [1:0]	[88:87]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [23:19]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

The two sets of 24-bit colors (Color 0,1 and Color 2,3) are then used to create two tables of four colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color2,3 table used for texels 16-31 indices. The color at index 1 is the linear interpolation of the base colors, while the color at index 3 is defined as Black (0,0,0) with Alpha = 0, as shown in the following figures:

FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 0-15)

Texel 0-15 Select	Color RGB	Alpha
0	Color0.RGB	0FFh
1	(Color0.RGB + Color1.RGB) /2	0FFh
2	Color1.RGB	0FFh
3	Black (0,0,0)	0

FXT CC_MIXED Interpolated Color Table (Alpha[0]=1, Texels 16-31)

Texel 16-31 Select	Color RGB	Alpha
0	Color2.RGB	0FFh
1	(Color2.RGB + Color3.RGB) /2	0FFh
2	Color3.RGB	0FFh
3	Black (0,0,0)	0

These tables are then used as a 4-entry Lookup Table, where each 2-bit Texel n Select field of the encoded CC_MIXED block is used to index into the appropriate 32-bit A8R8G8B8 color from the table, completing the decode of the CC_CMIXED block.

FXT1 CC_ALPHA Format

In the CC_ALPHA encoding format, three A5R5G5B5 colors are provided in the encoded block. A control bit (LERP) is used to define the lookup table (or tables) used to dereference the 2-bit Texel Selects.

CC_ALPHA Block Encoding

The following table describes the encoding of the 128-bit (DQWord) CC_ALPHA block format:

FXT CC_ALPHA Block Encoding

Bit	Description
127:125	Mode = '011'b (CC_ALPHA)
124	LERP
123:119	Color 2 Alpha
118:114	Color 1 Alpha
113:109	Color 0 Alpha
108:104	Color 2 Red
103:99	Color 2 Green
98:94	Color 2 Blue
93:89	Color 1 Red
88:84	Color 1 Green
83:79	Color 1 Blue
78:74	Color 0 Red
73:69	Color 0 Green
68:64	Color 0 Blue
63:62	Texel 31 Select
...	...
33:32	Texel 16 Select
31:30	Texel 15 Select
...	...
1:0	Texel 0 Select

CC_ALPHA Block Decoding

Each of the three colors (Color 0-2) are converted from A5R5G5B5 to A8R8G8B8 by replicating the 3 MSBs into the 3 LSBs, as shown in the following tables:

FXT CC_ALPHA Decoded Colors

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 2 [31:27]	Color 2 Alpha [7:3]	[123:119]
Color 2 [26:24]	Color 2 Alpha [2:0]	[123:121]
Color 2 [23:17]	Color 2 Red [7:3]	[108:104]
Color 2 [18:16]	Color 2 Red [2:0]	[108:106]
Color 2 [15:11]	Color 2 Green [7:3]	[103:99]
Color 2 [10:08]	Color 2 Green [2:0]	[103:101]
Color 2 [07:03]	Color 2 Blue [7:3]	[98:94]

Expanded Color Bit	Expanded Channel Bit	Encoded Block Source Bit
Color 2 [02:00]	Color 2 Blue [2:0]	[98:96]
Color 1 [31:27]	Color 1 Alpha [7:3]	[118:114]
Color 1 [26:24]	Color 1 Alpha [2:0]	[118:116]
Color 1 [23:17]	Color 1 Red [7:3]	[93:89]
Color 1 [18:16]	Color 1 Red [2:0]	[93:91]
Color 1 [15:11]	Color 1 Green [7:3]	[88:84]
Color 1 [10:08]	Color 1 Green [2:0]	[88:86]
Color 1 [07:03]	Color 1 Blue [7:3]	[83:79]
Color 1 [02:00]	Color 1 Blue [2:0]	[83:81]
Color 0 [31:27]	Color 0 Alpha [7:3]	[113:109]
Color 0 [26:24]	Color 0 Alpha [2:0]	[113:111]
Color 0 [23:17]	Color 0 Red [7:3]	[78:74]
Color 0 [18:16]	Color 0 Red [2:0]	[78:76]
Color 0 [15:11]	Color 0 Green [7:3]	[73:69]
Color 0 [10:08]	Color 0 Green [2:0]	[73:71]
Color 0 [07:03]	Color 0 Blue [7:3]	[68:64]
Color 0 [02:00]	Color 0 Blue [2:0]	[68:66]

LERP = 0 Decoding

When LERP = 0, a single 4-entry lookup table is formed using the three expanded colors, with the 4th entry defined as transparent black (ARGB=0,0,0,0). Each 2-bit Texel n Select field of the encoded CC_ALPHA block is used to index into a 32-bit A8R8G8B8 color from the table completing the decode of the CC_ALPHA block.

FXT CC_ALPHA Interpolated Color Table (LERP=0)

Texel Select	Color	Alpha
0	Color0.RGB	Color0.Alpha
1	Color1.RGB	Color1.Alpha
2	Color2.RGB	Color2.Alpha
3	Black (RGB=0,0,0)	0

LERP = 1 Decoding

When LERP = 1, the three expanded colors are used to create two tables of four interpolated colors. The Color0,1 table is used as a lookup table for texel 0-15 indices, and the Color1,2 table used for texels 16-31 indices, as shown in the following figures:

FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 0-15)

Texel 0-15 Select	Color ARGB
0	Color0.ARGB
1	(2*Color0.ARGB + Color1.ARGB + 1) /3
2	(Color0.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

FXT CC_ALPHA Interpolated Color Table (LERP=1, Texels 16-31)

Texel 16-31 Select	Color ARGB
0	Color2.ARGB
1	(2*Color2.ARGB + Color1.ARGB + 1) /3
2	(Color2.ARGB + 2*Color1.ARGB + 1) /3
3	Color1.ARGB

DXT/BC1-3 Texture Formats

Note that non-power-of-2 dimensioned maps may require the surface to be padded out to the next multiple of four texels – here the pad texels are not referenced by the device.

An 8-byte (QWord) block encoding can be used if the source texture contains no transparency (is opaque) or if the transparency can be specified by a one-bit alpha. A 16-byte (DQWord) block encoding can be used to support source textures that require more than one-bit alpha: here the 1st QWord is used to encode the texel alpha values, and the 2nd QWord is used to encode the texel color values.

These three types of format are discussed in the following sections:

- Opaque and One-bit Alpha Textures (DXT1)
- Opaque Textures (DXT1_RGB)
- Textures with Alpha Channels (DXT2-5)

DXT2 and DXT3 are equivalent compression formats from the perspective of the hardware. The only difference between the two is the use of pre-multiplied alpha encoding, which does not affect hardware.

Likewise, DXT4 and DXT5 are the same compression formats with the only difference being the use of pre-multiplied alpha encoding.

Note that the surface formats DXT1-5 are referred to in the DirectX Specification as BC1-3. The mapping between formats is shown below:

- DXT1 ⇒ BC1
- DXT2/DXT3 ⇒ BC2
- DXT4/DXT5 ⇒ BC3

Programming Note	
Context:	DXT Texture Formats
• Any single texture must specify that its data is stored as 64 or 128 bits per group of 16 texels. If 64-bit blocks—that is, format DXT1—are used for the texture, it is possible to mix the opaque and one-bit alpha formats on a per-block basis within the same texture. In other words, the comparison of the unsigned integer magnitude of color_0 and color_1 is performed uniquely for each block of 16 texels. • When 128-bit blocks are used, then the alpha channel must be specified in either explicit (format DXT2 or DXT3) or interpolated mode (format DXT4 or DXT5) for the entire texture. Note that as with color, once interpolated mode is selected then either 8 interpolated alphas or 6 interpolated alphas mode can be used on a block-by-block basis. Again the magnitude comparison of alpha_0 and alpha_1 is done uniquely on a block-by-block basis.	

Opaque and One-bit Alpha Textures (DXT1/BC1)

Texture format DXT1 is for textures that are opaque or have a single transparent color. For each opaque or one-bit alpha block, two 16-bit R5G6B5 values and a 4x4 bitmap with 2-bits-per-pixel are stored. This totals 64 bits (1 QWord) for 16 texels, or 4-bits-per-texel.

In the block bitmap, there are two bits per texel to select between the four colors, two of which are stored in the encoded data. The other two colors are derived from these stored colors by linear interpolation.

The one-bit alpha format is distinguished from the opaque format by comparing the two 16-bit color values stored in the block. They are treated as unsigned integers. If the first color is greater than the second, it implies that only opaque texels are defined. This means four colors will be used to represent the texels. In four-color encoding, there are two derived colors and all four colors are equally distributed in RGB color space. This format is analogous to R5G6B5 format. Otherwise, for one-bit alpha transparency, three colors are used and the fourth is reserved to represent transparent texels. Note that the color blocks in DXT2-5 formats strictly use four colors, as the alpha values are obtained from the alpha block .

In three-color encoding, there is one derived color and the fourth two-bit code is reserved to indicate a transparent texel (alpha information). This format is analogous to A1R5G5B5, where the final bit is used for encoding the alpha mask.

The following piece of pseudo-code illustrates the algorithm for deciding whether three- or four-color encoding is selected:

```
 if (color_0 > color_1)
 {
   // Four-color block: derive the other two colors.
   // 00 = color_0, 01 = color_1, 10 = color_2, 11 = color_3
   // These two bit codes correspond to the 2-bit fields
   // stored in the 64-bit block.  color_2 = (2 * color_0 + color_1) / 3;
    color_3 = (color 0 + 2 * color_1) / 3;
 }
 else
 {
   // Three-color block: derive the other color.  // 00 = color_0, 01 = color_1, 10 =
color_2,
   // 11 = transparent.  // These two bit codes correspond to the 2-bit fields
   // stored in the 64-bit block.  color_2 = (color_0 + color_1) / 2;
    color_3 = transparent;
 }
```

The following tables show the memory layout for the 8-byte block. It is assumed that the first index corresponds to the y-coordinate and the second corresponds to the x-coordinate. For example, Texel[1][2] refers to the texture map pixel at (x,y) = (2,1).

Here is the memory layout for the 8-byte (64-bit) block:

Word Address	16-bit Word
0	Color_0
1	Color_1
2	Bitmap Word_0
3	Bitmap Word_1

Color_0 and Color_1 (colors at the two extremes) are laid out as follows:

Bits	Color
15:11	Red color component
10:5	Green color component
4:0	Blue color component

Bits	Texel
1:0 (LSB)	Texel[0][0]
3:2	Texel[0][1]
5:4	Texel[0][2]
7:6	Texel[0][3]
9:8	Texel[1][0]
11:10	Texel[1][1]
13:12	Texel[1][2]
15:14	Texel[1][3]

Bitmap Word_1 is laid out as follows:

Bits	Texel
1:0 (LSB)	Texel[2][0]
3:2	Texel[2][1]
5:4	Texel[2][2]
7:6	Texel[2][3]
9:8	Texel[3][0]
11:10	Texel[3][1]
13:12	Texel[3][2]
15:14 (MSB)	Texel[3][3]

Example of Opaque Color Encoding

As an example of opaque encoding, we will assume that the colors red and black are at the extremes. We will call red color_0 and black color_1. There will be four interpolated colors that form the uniformly distributed gradient between them. To determine the values for the 4x4 bitmap, the following calculations are used:

```
00 ? color_0
01 ? color_1
10 ? 2/3 color_0 + 1/3 color_1
11 ? 1/3 color_0 + 2/3 color_1
```

Example of One-bit Alpha Encoding

This format is selected when the unsigned 16-bit integer, color_0, is less than the unsigned 16-bit integer, color_1. An example of where this format could be used is leaves on a tree to be shown against a blue sky. Some texels could be marked as transparent while three shades of green are still available for the leaves. Two of these colors fix the extremes, and the third color is an interpolated color.

The bitmap encoding for the colors and the transparency is determined using the following calculations:

```
00 ? color_0
01 ? color_1
10 ? 1/2 color_0 + 1/2 color_1
11 ? Transparent
```

Opaque Textures (DXT1_RGB)

Texture format DXT1_RGB is identical to DXT1, with the exception that the One-bit Alpha encoding is removed. Color 0 and Color 1 are not compared, and the resulting texel color is derived strictly from the Opaque Color Encoding. The alpha channel defaults to 1.0.

Programming Note	
Context:	Opaque Textures (DXT1_RGB)
The behavior of this format is not compliant with the OGL spec.	

Compressed Textures with Alpha Channels (DXT2-5 / BC2-3)

There are two ways to encode texture maps that exhibit more complex transparency. In each case, a block that describes the transparency precedes the 64-bit block already described for DXT1. The transparency is either represented as a 4x4 bitmap with four bits per pixel (explicit encoding), or with fewer bits and linear interpolation analogous to what is used for color encoding.

The transparency block and the color block are laid out as follows:

Word Address	64-bit Block
3:0	Transparency block
7:4	Previously described 64-bit block

Explicit Texture Encoding

For explicit texture encoding (DXT2 and DXT3 formats), the alpha components of the texels that describe transparency are encoded in a 4x4 bitmap with 4 bits per texel. These 4 bits can be achieved through a variety of means such as dithering or by simply using the 4 most significant bits of the alpha data. However they are produced, they are used just as they are, without any form of interpolation.

Note: DirectDraw's compression method uses the 4 most significant bits.

The following tables illustrate how the alpha information is laid out in memory, for each 16-bit word.

This is the layout for Word 0:

Bits	Alpha
3:0 (LSB)	[0][0]
7:4	[0][1]
11:8	[0][2]
15:12 (MSB)	[0][3]

This is the layout for Word 1:

Bits	Alpha
3:0 (LSB)	[1][0]
7:4	[1][1]
11:8	[1][2]
15:12 (MSB)	[1][3]

This is the layout for Word 2:

Bits	Alpha
3:0 (LSB)	[2][0]
7:4	[2][1]
11:8	[2][2]
15:12 (MSB)	[2][3]

This is the layout for Word 3:

Bits	Alpha
3:0 (LSB)	[3][0]
7:4	[3][1]
11:8	[3][2]
15:12 (MSB)	[3][3]

Three-Bit Linear Alpha Interpolation

The encoding of transparency for the DXT4 and DXT5 formats is based on a concept similar to the linear encoding used for color. Two 8-bit alpha values and a 4x4 bitmap with three bits per pixel are stored in the first eight bytes of the block. The representative alpha values are used to interpolate intermediate alpha values. Additional information is available in the way the two alpha values are stored. If alpha_0 is greater than alpha_1, then six intermediate alpha values are created by the interpolation. Otherwise, four intermediate alpha values are interpolated between the specified alpha extremes. The two additional implicit alpha values are 0 (fully transparent) and 255 (fully opaque).

The following pseudo-code illustrates this algorithm:

```
// 8-alpha or 6-alpha block?
if (alpha_0 > alpha_1) {
   // 8-alpha block: derive the other 6 alphas.
   // 000 = alpha_0, 001 = alpha_1, others are interpolated
  alpha_2 = (6 * alpha_0 + alpha_1) / 7;      // Bit code 010
  alpha_3 = (5 * alpha_0 + 2 * alpha_1) / 7; // Bit code 011
  alpha_4 = (4 * alpha_0 + 3 * alpha_1) / 7; // Bit code 100
  alpha_5 = (3 * alpha_0 + 4 * alpha_1) / 7; // Bit code 101
  alpha_6 = (2 * alpha_0 + 5 * alpha_1) / 7; // Bit code 110
  alpha_7 = (alpha_0 + 6 * alpha_1) / 7;      // Bit code 111
 }
else {
   // 6-alpha block: derive the other alphas.
   // 000 = alpha_0, 001 = alpha_1, others are interpolated
  alpha_2 = (4 * alpha_0 + alpha_1) / 5;      // Bit code 010
  alpha_3 = (3 * alpha_0 + 2 * alpha_1) / 5; // Bit code 011
  alpha_4 = (2 * alpha_0 + 3 * alpha_1) / 5; // Bit code 100
  alpha_5 = (alpha_0 + 4 * alpha_1) / 5;      // Bit code 101
  alpha_6 = 0;                                // Bit code 110
  alpha_7 = 255;                              // Bit code 111
}
```

The memory layout of the alpha block is as follows:

Byte	Alpha
0	Alpha_0
1	Alpha_1
2	[0][2] (2 LSBs), [0][1], [0][0]
3	[1][1] (1 LSB), [1][0], [0][3], [0][2] (1 MSB)
4	[1][3], [1][2], [1][1] (2 MSBs)
5	[2][2] (2 LSBs), [2][1], [2][0]
6	[3][1] (1 LSB), [3][0], [2][3], [2][2] (1 MSB)
7	[3][3], [3][2], [3][1] (2 MSBs)

BC4

These formats (BC4_UNORM and BC4_SNORM) compresses single-component UNORM or SNORM data. An 8-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 8-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] bit code
21:19	texel[0][1] bit code
24:22	texel[0][2] bit code
27:25	texel[0][3] bit code
30:28	texel[1][0] bit code
33:31	texel[1][1] bit code
36:34	texel[1][2] bit code
39:37	texel[1][3] bit code
42:40	texel[2][0] bit code
45:43	texel[2][1] bit code
48:46	texel[2][2] bit code
51:49	texel[2][3] bit code
54:52	texel[3][0] bit code
57:55	texel[3][1] bit code
60:58	texel[3][2] bit code
63:61	texel[3][3] bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                          // bit code 000
red_1 = red_1;                          // bit code 001
if (red_0 > red_1) {
    red_2 = (6 * red_0 + 1 * red_1) / 7; // bit code 010
    red_3 = (5 * red_0 + 2 * red_1) / 7; // bit code 011
    red_4 = (4 * red_0 + 3 * red_1) / 7; // bit code 100
    red_5 = (3 * red_0 + 4 * red_1) / 7; // bit code 101
    red_6 = (2 * red_0 + 5 * red_1) / 7; // bit code 110
    red_7 = (1 * red_0 + 6 * red_1) / 7; // bit code 111
}
else {
    red_2 = (4 * red_0 + 1 * red_1) / 5; // bit code 010
    red_3 = (3 * red_0 + 2 * red_1) / 5; // bit code 011
    red_4 = (2 * red_0 + 3 * red_1) / 5; // bit code 100
    red_5 = (1 * red_0 + 4 * red_1) / 5; // bit code 101
    red_6 = UNORM ? 0.0 : -1.0;          // bit code 110 (0 for UNORM, -1 for SNORM)
    red_7 = 1.0;                         // bit code 111
}
```

BC5

These formats (BC5_UNORM and BC5_SNORM) compresses dual-component UNORM or SNORM data. A 16-byte compression block represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel.

The 16-byte compression block is laid out as follows:

Bit	Description
7:0	red_0
15:8	red_1
18:16	texel[0][0] red bit code
21:19	texel[0][1] red bit code
24:22	texel[0][2] red bit code
27:25	texel[0][3] red bit code
30:28	texel[1][0] red bit code
33:31	texel[1][1] red bit code
36:34	texel[1][2] red bit code
39:37	texel[1][3] red bit code
42:40	texel[2][0] red bit code
45:43	texel[2][1] red bit code
48:46	texel[2][2] red bit code
51:49	texel[2][3] red bit code

Bit	Description
54:52	texel[3][0] red bit code
57:55	texel[3][1] red bit code
60:58	texel[3][2] red bit code
63:61	texel[3][3] red bit code
71:64	green_0
79:72	green_1
82:80	texel[0][0] green bit code
85:83	texel[0][1] green bit code
88:86	texel[0][2] green bit code
91:89	texel[0][3] green bit code
94:92	texel[1][0] green bit code
97:95	texel[1][1] green bit code
100:98	texel[1][2] green bit code
103:101	texel[1][3] green bit code
106:104	texel[2][0] green bit code
109:107	texel[2][1] green bit code
112:110	texel[2][2] green bit code
115:113	texel[2][3] green bit code
118:116	texel[3][0] green bit code
121:119	texel[3][1] green bit code
124:122	texel[3][2] green bit code
127:125	texel[3][3] green bit code

There are two interpolation modes, chosen based on which reference color is larger. The first mode has the two reference colors plus six equal-spaced interpolated colors between the reference colors, chosen based on the three-bit code for that texel. The second mode has the two reference colors plus four interpolated colors, chosen by six of the three-bit codes. The remaining two codes select min and max values for the colors. The values of red_0 through red_7 are computed as follows:

```
red_0 = red_0;                          // bit code 000
red_1 = red_1;                          // bit code 001
if (red_0 > red_1) {
    red_2 = (6 * red_0 + 1 * red_1) / 7; // bit code 010
    red_3 = (5 * red_0 + 2 * red_1) / 7; // bit code 011
    red_4 = (4 * red_0 + 3 * red_1) / 7; // bit code 100
    red_5 = (3 * red_0 + 4 * red_1) / 7; // bit code 101
    red_6 = (2 * red_0 + 5 * red_1) / 7; // bit code 110
    red_7 = (1 * red_0 + 6 * red_1) / 7; // bit code 111
}
else {
    red_2 = (4 * red_0 + 1 * red_1) / 5; // bit code 010
    red_3 = (3 * red_0 + 2 * red_1) / 5; // bit code 011
    red_4 = (2 * red_0 + 3 * red_1) / 5; // bit code 100
    red_5 = (1 * red_0 + 4 * red_1) / 5; // bit code 101
    red_6 = UNORM ? 0.0 : -1.0;          // bit code 110 (0 for UNORM, -1 for SNORM)
    red_7 = 1.0;                         // bit code 111
}
```

The same calculations are done for green, using the corresponding reference colors and bit codes.

BC6H

For CHV, BSW, these formats (BC6H_UF16 and BC6H_SF16) compresses 3-channel images with high dynamic range (> 8 bits per channel). BC6H supports floating point denorms but there is no support for INF and NaN, other than with BC6H_SF16 –INF is supported. The alpha channel is not included, thus alpha is returned at its default value.

The BC6H block is 16 bytes and represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel. BC6H has 14 different modes, the mode that the block is in is contained in the least significant bits (either 2 or 5 bits).

The basic scheme consists of interpolating colors along either one or two lines, with per-texel indices indicating which color along the line is chosen for each texel. If a two-line mode is selected, one of 32 partition sets is indicated which selects which of the two lines each texel is assigned to.

Field Definition

There are 14 possible modes for a BC6H block, the format of each is indicated in the 14 tables below. The mode is selected by the unique mode bits specified in each table. The first 10 modes use two lines (“TWO”), and the last 4 use one line (“ONE”). The difference between the various two-line and one-line modes is with the precision of the first endpoint and the number of bits used to store delta values for the remaining endpoints. Two modes (9 and 10) specify each endpoint as an original value rather than using the deltas (these are indicated as having no delta values).

The endpoints values and deltas are indicated in the tables using a two-letter name. The first letter is “r”, “g”, or “b” indicating the color channel. The second letter is “w”, “x”, “y”, or “z” indicating which of the four endpoints. The first line has endpoints “w” and “x”, with “w” being the endpoint that is fully specified (i.e. not as a delta). The second line has endpoints “y” and “z”. Modes using ONE mode do not have endpoints “y” and “z” as they have only one line.

In addition to the mode and endpoint data, TWO blocks contain a 5-bit “partition” which selects one of the partition sets, and a 46-bit set of indices. ONE blocks contain a 63-bit set of indices. These are described in more detail below.

Mode 0: (TWO) Red, Green, Blue: 10-bit endpoint, 5-bit deltas

Bit	Description
1:0	mode = 00
2	gy[4]
3	by[4]
4	bz[4]
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 1: (TWO) Red, Green, Blue: 7-bit endpoint, 6-bit deltas

Bit	Description
1:0	mode = 01
2	gy[5]
3	gz[4]
4	gz[5]
11:5	rw[6:0]
12	bz[0]
13	bz[1]
14	by[4]
21:15	gw[6:0]
22	by[5]
23	bz[2]
24	gy[4]
31:25	bw[6:0]
32	bz[3]
33	bz[5]
34	bz[4]
40:35	rx[5:0]
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]
60:55	bx[5:0]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 2: (TWO) Red: 11-bit endpoint, 5-bit deltas

Green, Blue: 11-bit endpoint, 4-bit deltas

Bit	Description
4:0	mode = 00010
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
39:35	rx[4:0]
40	rw[10]
44:41	gy[3:0]
48:45	gx[3:0]
49	gw[10]
50	bz[0]
54:51	gz[3:0]
58:55	bx[3:0]
59	bw[10]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 3: (TWO) Red, Blue: 11-bit endpoint, 4-bit deltas

Green: 11-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 00110
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[10]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	gw[10]
54:51	gz[3:0]
58:55	bx[3:0]
59	bw[10]
60	bz[1]
64:61	by[3:0]
68:65	ry[3:0]
69	bz[0]
70	bz[2]
74:71	rz[3:0]
75	gy[4]
76	bz[3]
81:77	partition
127:82	indices

Mode 4: (TWO) Red, Green: 11-bit endpoint, 4-bit deltas

Blue: 11-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 01010
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[10]
40	by[4]
44:41	gy[3:0]
48:45	gx[3:0]
49	gw[10]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bw[10]
64:61	by[3:0]
68:65	ry[3:0]
69	bz[1]
70	bz[2]
74:71	rz[3:0]
75	bz[4]
76	bz[3]
81:77	partition
127:82	indices

Mode 5: (TWO) Red, Green, Blue: 9-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 01110
13:5	rw[8:0]
14	by[4]
23:15	gw[8:0]
24	gy[4]
33:25	bw[8:0]
34	bz[4]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
49:45	gx[3:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 6: (TWO) Red: 8-bit endpoint, 6-bit deltas

Green, Blue: 8-bit endpoint, 5-bit deltas

Bit	Description
4:0	mode = 10010
12:5	rw[7:0]
13	gz[4]
14	by[4]
22:15	gw[7:0]
23	bz[2]
24	gy[4]
32:25	bw[7:0]
33	bz[3]
34	bz[4]
40:35	rx[5:0]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
59:55	bx[4:0]
60	gz[1]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 7: (TWO) Red, Blue: 8-bit endpoint, 5-bit deltas

Green: 8-bit endpoint, 6-bit deltas

Bit	Description
4:0	mode = 10110
12:5	rw[7:0]
13	bz[0]
14	by[4]
22:15	gw[7:0]
23	gy[5]
24	gy[4]
32:25	bw[7:0]
33	gz[5]
34	bz[4]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]
59:55	bx[4:0]
60	bz[1]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 8: (TWO) Red, Green: 8-bit endpoint, 5-bit deltas

Blue: 8-bit endpoint, 6-bit deltas

Bit	Description
4:0	mode = 11010
12:5	rw[7:0]
13	bz[1]
14	by[4]
22:15	gw[7:0]
23	by[5]
24	gy[4]
32:25	bw[7:0]
33	bz[5]
34	bz[4]
39:35	rx[4:0]
40	gz[4]
44:41	gy[3:0]
49:45	gx[4:0]
50	bz[0]
54:51	gz[3:0]
60:55	bx[5:0]
64:61	by[3:0]
69:65	ry[4:0]
70	bz[2]
75:71	rz[4:0]
76	bz[3]
81:77	partition
127:82	indices

Mode 9: (TWO) Red, Green, Blue: 6-bit endpoints for all four, no deltas

Bit	Description
4:0	mode = 11110
10:5	rw[5:0]
11	gz[4]
12	bz[0]
13	bz[1]
14	by[4]
20:15	gw[5:0]
21	gy[5]
22	by[5]
23	bz[2]
24	gy[4]
30:25	bw[5:0]
31	gz[5]
32	bz[3]
33	bz[5]
34	bz[4]
40:35	rx[5:0]
44:41	gy[3:0]
50:45	gx[5:0]
54:51	gz[3:0]
60:55	bx[5:0]
64:61	by[3:0]
70:65	ry[5:0]
76:71	rz[5:0]
81:77	partition
127:82	indices

Mode 10: (ONE) Red, Green, Blue: 10-bit endpoints for both, no deltas

Bit	Description
4:0	mode = 00011
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
44:35	rx[9:0]
54:45	gx[9:0]
64:55	bx[9:0]
127:65	indices

Mode 11: (ONE) Red, Green, Blue: 11-bit endpoints, 9-bit deltas

Bit	Description
4:0	mode = 00111
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
43:35	rx[8:0]
44	rw[10]
53:45	gx[8:0]
54	gw[10]
63:55	bx[8:0]
64	bw[10]
127:65	indices

Mode 12: (ONE) Red, Green, Blue: 12-bit endpoints, 8-bit deltas

Bit	Description
4:0	mode = 01011
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
42:35	rx[7:0]
43	rw[11]
44	rw[10]
52:45	gx[7:0]
53	gw[11]
54	gw[10]
62:55	bx[7:0]
63	bw[11]
64	bw[10]
127:65	indices

Mode 13: (ONE) Red, Green, Blue: 16-bit endpoints, 4-bit deltas

Bit	Description
4:0	mode = 01111
14:5	rw[9:0]
24:15	gw[9:0]
34:25	bw[9:0]
38:35	rx[3:0]
39	rw[15]
40	rw[14]
41	rw[13]
42	rw[12]
43	rw[11]
44	rw[10]
48:45	gx[3:0]
49	gw[15]
50	gw[14]
51	gw[13]
52	gw[12]
53	gw[11]
54	gw[10]
58:55	bx[3:0]
59	bw[15]
60	bw[14]
61	bw[13]
62	bw[12]
63	bw[11]
64	bw[10]
127:65	indices

Undefined mode values (10011, 10111, 11011, and 11111) return zero in the RGB channels.

The “indices” fields are defined as follows:

TWO mode *indices* field with fix-up index [1] at texel[3][3]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
89:87	texel[0][2] index
92:90	texel[0][3] index
95:93	texel[1][0] index
98:96	texel[1][1] index
101:99	texel[1][2] index
104:102	texel[1][3] index
107:105	texel[2][0] index
110:108	texel[2][1] index
113:111	texel[2][2] index
116:114	texel[2][3] index
119:117	texel[3][0] index
122:120	texel[3][1] index
125:123	texel[3][2] index
127:126	texel[3][3] index

TWO mode *indices* field with fix-up index [1] at texel[0][2]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
88:87	texel[0][2] index
91:89	texel[0][3] index
94:92	texel[1][0] index
97:95	texel[1][1] index
100:98	texel[1][2] index
103:101	texel[1][3] index
106:104	texel[2][0] index
109:107	texel[2][1] index
112:110	texel[2][2] index
115:113	texel[2][3] index
118:116	texel[3][0] index
121:119	texel[3][1] index
124:122	texel[3][2] index
127:125	texel[3][3] index

TWO mode *indices* field with fix-up index [1] at texel[2][0]

Bit	Description
83:82	texel[0][0] index
86:84	texel[0][1] index
89:87	texel[0][2] index
92:90	texel[0][3] index
95:93	texel[1][0] index
98:96	texel[1][1] index
101:99	texel[1][2] index
104:102	texel[1][3] index
106:105	texel[2][0] index
109:107	texel[2][1] index
112:110	texel[2][2] index
115:113	texel[2][3] index
118:116	texel[3][0] index
121:119	texel[3][1] index
124:122	texel[3][2] index
127:125	texel[3][3] index

ONE mode *indices* field

Bit	Description
67:65	texel[0][0] index
71:68	texel[0][1] index
75:72	texel[0][2] index
79:76	texel[0][3] index
83:80	texel[1][0] index
87:84	texel[1][1] index
91:88	texel[1][2] index
95:92	texel[1][3] index
99:96	texel[2][0] index
103:100	texel[2][1] index
107:104	texel[2][2] index
111:108	texel[2][3] index
115:112	texel[3][0] index
119:116	texel[3][1] index
123:120	texel[3][2] index

Bit	Description
127:124	texel[3][3] index

Endpoint Computation

The endpoints can be defined in many different ways, as shown above. This section describes how the endpoints are computed from the bits in the compression block. The method used depends on whether the BC6H format is signed (BC6H_SF16) or unsigned (BC6H_UF16).

First, each channel (RGB) of each endpoint is extended to 16 bits. Each is handled identically and independently, however in some modes different channels have different incoming precision which must be accounted for. The following rules are employed:

- If the format is BC6H_SF16 or the endpoint is a delta value, the value is sign-extended to 16 bits
- For all other cases, the value is zero-extended to 16 bits

If there are no endpoints that are delta values, endpoint computation is complete. For endpoints that are delta values, the next step involves computing the absolute endpoint. The “w” endpoint is always absolute and acts as a base value for the other three endpoints. Each channel is handled identically and independently.

```
x = w + x
y = w + y
z = w + z
```

The above is performed using 16-bit integer arithmetic. Overflows beyond 16 bits are ignored (any resulting high bits are dropped).

Palette Color Computation

The next step involves computing the color palette values that provide the available values for each texel's color. The color palette for each line consists of the two endpoint colors plus 6 (TWO mode) or 14 (ONE mode) interpolated colors. Again each channel is processed independently.

First the endpoints are unquantized, with each channel of each endpoint being processed independently. The number of bits in the original base *w* value represents the precision of the endpoints. The input endpoint is called *e*, and the resulting endpoints are represented as 17-bit signed integers and called e' below.

For the BC6H_UF16 format:

- if the precision is already 16 bits, e' = e
- if e = 0, e' = 0
- if e is the maximum representible in the precision, e' = 0xFFFF
- otherwise, e' = ((e « 16) + 0x8000) » precision

For the BC6H_SF16 format, the value is treated as sign magnitude. The sign is not changed, e' and e refer only to the magnitude portion:

- if the precision is already 16 bits, e' = e
- if e = 0, e' = 0
- if e is the maximum representible in the precision, e' = 0x7FFF
- otherwise, e' = ((e « 15) + 0x4000) » (precision - 1)

Next, the palette values are generated using predefined weights, using the tables below:

```
palette[i] = (w' * (64 - weight[i]) + x' * weight[i] + 32) » 6
```

TWO mode weights:

palette index	0	1	2	3	4	5	6	7
weight	0	9	18	27	37	46	55	64

ONE mode weights:

palette index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
weight	0	4	9	13	17	21	26	30	34	38	43	47	51	55	60	64

The two end palette indices are equal to the two endpoints given that the weights are 0 and 64. In the above equation w' and x' represent the endpoints e' computed in the previous step corresponding to w and x, respectively. For the second line in TWO mode, w and x are replaced with y and z.

The final step in computing the palette colors is to rescale the final results. For BC6H_UF16 format, the values are multiplied by 31/64. For BC6H_SF16, the values are multiplied by 31/32, treating them as sign magnitude. These final 16-bit results are ultimately treated as 16-bit floats.

Texel Selection

The final step is to select the appropriate palette index for each texel. This index then selects the 16-bit per channel palette value, which is re-interpreted as a 16-bit floating point result for input into the filter. This procedure differs depending on whether the mode is TWO or ONE.

ONE Mode

In ONE mode, there is only one set of palette colors, but the “indices” field is 63 bits. This field consists of a 4-bit palette index for each of the 16 texels, with the exception of the texel at [0][0] which has only 3 bits, the missing high bit being set to zero.

TWO Mode

32 partitions are defined for TWO, which are defined below. Each of the 32 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-1C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *w* and *x*) or line 1

(endpoints *y* and *z*). Each case has one texel each of "[0]" and "[1]", the index that this is at is termed the "fix-up index". These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	1	[0]	0	0	1	[0]	1	1	1	[0]	0	0	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	0	0	0	[1]	0	1	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	1	[0]	0	0	1	[0]	0	0	0
	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
08	[0]	0	0	0	[0]	0	1	1	[0]	0	0	0	[0]	0	0	0
	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0
	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
0C	[0]	0	0	1	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]
10	[0]	0	0	0	[0]	1	[1]	1	[0]	0	0	0	[0]	1	[1]	1
	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	1	1	1	0	0	0	0	0	[1]	0	0	0	0	0	0	1
	1	1	1	[1]	0	0	0	0	1	1	1	0	0	0	0	0
14	[0]	0	[1]	1	[0]	0	0	0	[0]	0	0	0	[0]	1	1	1
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1
	0	0	0	0	[1]	1	0	0	[1]	0	0	0	0	0	1	1
	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	[1]
18	[0]	0	[1]	1	[0]	0	0	0	[0]	1	[1]	0	[0]	0	[1]	1
	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	1	[1]	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0
1C	[0]	0	0	1	[0]	0	0	0	[0]	1	[1]	1	[0]	0	[1]	1
	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1
	[1]	1	1	0	[1]	1	1	1	1	0	0	0	1	0	0	1
	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0

The 46-bit "indices" field consists of a 3-bit palette index for each of the 16 texels, with the exception of the bracketed texels that have only two bits each. The high bit of these texels is set to zero.

BC7

These formats (BC7_UNORM and BC7_UNORM_SRGB) compresses 3-channel and 4-channel fixed point images.

The BC7 block is 16 bytes and represents a 4x4 block of texels. The texels are labeled as texel[row][column] where both row and column range from 0 to 3. Texel[0][0] is the upper left texel. BC7 has 8 different modes, the mode that the block is in is contained in the least significant bits (1-8 bits depending on mode).

The basic scheme consists of interpolating colors and alpha in some modes along either one, two, or three lines, with per-texel indices indicating which color/alpha along the line is chosen for each texel. If a two- or three-line mode is selected, one of 64 partition sets is indicated which selects which of the two lines each texel is assigned to, although some modes are limited to the first 16 partition sets. In the color-only modes, alpha is always returned at its default value of 1.0.

Some modes contain the following fields:

- **P-bits.** These represent shared LSB for all components of the endpoint, which increases the endpoint precision by one bit. In some cases both endpoints of a line share a P-bit.
- **Rotation bits.** For blocks with separate color and alpha, this 2-bit field allows selection of which of the four components has its own indexes (scalar) vs. the other three components (vector).
- **Index selector.** This 1-bit field selects whether the scalar or vector components uses the 3-bit index vs. the 2-bit index.

Field Definition

There are 8 possible modes for a BC7 block, the format of each is indicated in the 8 tables below. The mode is selected by the unique mode bits specified in each table. Each mode has particular characteristics described at the top of the table.

Mode 0: Color only, 3 lines (THREE), 4-bit endpoints with one P-bit per endpoint, 3-bit indices, 16 partitions

Bit	Description
0	mode = 0
4:1	partition
8:5	R0
12:9	R1
16:13	R2
20:17	R3
24:21	R4
28:25	R5
32:29	G0

Bit	Description
36:33	G1
40:37	G2
44:41	G3
48:45	G4
52:49	G5
56:53	B0
60:57	B1
64:61	B2
68:65	B3
72:69	B4
76:73	B5
77	P0
78	P1
79	P2
80	P3
81	P4
82	P5
127:83	indices

Mode 1: Color only, 2 lines (TWO), 6-bit endpoints with one shared P-bit per line, 3-bit indices, 64 partitions

Bit	Description
1:0	mode = 10
7:2	partition
13:8	R0
19:14	R1
25:20	R2
31:26	R3
37:32	G0
43:38	G1
49:44	G2
55:50	G3
61:56	B0
67:62	B1
73:68	B2
79:74	B3
80	P0
81	P1
127:82	indices

Mode 2: Color only, 3 lines (THREE), 5-bit endpoints, 2-bit indices, 64 partitions

Bit	Description
2:0	mode = 100
8:3	partition
13:9	R0
18:14	R1
23:19	R2
28:24	R3
33:29	R4
38:34	R5
43:39	G0
48:44	G1
53:49	G2
58:54	G3
63:59	G4
68:64	G5
73:69	B0
78:74	B1
83:79	B2
88:84	B3
93:89	B4
98:94	B5
127:99	indices

Mode 3: Color only, 2 lines (TWO), 7-bit endpoints with one P-bit per endpoint, 2-bit indices, 64 partitions

Bit	Description
3:0	mode = 1000
9:4	partition
16:10	R0
23:17	R1
30:24	R2
37:31	R3
44:38	G0
51:45	G1
58:52	G2
65:59	G3
72:66	B0
79:73	B1
86:80	B2
93:87	B3
94	P0
95	P1
96	P2
97	P3
127:98	indices

Mode 4: Color and alpha, 1 line (ONE), 5-bit color endpoints, 6-bit alpha endpoints, 16 2-bit indices, 16 3-bit indices, 2-bit component rotation, 1-bit index selector

Bit	Description
4:0	mode = 10000
6:5	rotation
7	index selector
12:8	R0
17:13	R1
22:18	G0
27:23	G1
32:28	B0
37:33	B1
43:38	A0
49:44	A1

Bit	Description
80:50	2-bit indices
127:81	3-bit indices

Mode 5: Color and alpha, 1 line (ONE), 7-bit color endpoints, 8-bit alpha endpoints, 2-bit color indices, 2-bit alpha indices, 2-bit component rotation

Bit	Description
5:0	mode = 100000
7:6	rotation
14:8	R0
21:15	R1
28:22	G0
35:29	G1
42:36	B0
49:43	B1
57:50	A0
65:58	A1
96:66	color indices
127:97	alpha indices

Mode 6: Combined color and alpha, 1 line (ONE), 7-bit endpoints with one P-bit per endpoint, 4-bit indices

Bit	Description
6:0	mode = 1000000
13:7	R0
20:14	R1
27:21	G0
34:28	G1
41:35	B0
48:42	B1
55:49	A0
62:56	A1
63	P0
64	P1
127:65	indices

Mode 7: Combined color and alpha, 2 lines (TWO), 5-bit endpoints with one P-bit per endpoint, 2-bit indices, 64 partitions

Bit	Description
7:0	mode = 10000000
13:8	partition
18:14	R0
23:19	R1
28:24	R2
33:29	R3
38:34	G0
43:39	G1
48:44	G2
53:49	G3
58:54	B0
63:59	B1
68:64	B2
73:69	B3
78:74	A0
83:79	A1
88:84	A2
93:89	A3
94	P0
95	P1
96	P2
97	P3
127:98	indices

Undefined mode values (bits 7:0 = 00000000) return zero in the RGB channels.

The indices fields are variable in length and due to the different locations of the fix-up indices depending on partition set there are a very large number of possible configurations. Each mode above indicates how many bits each index has, and the fix-up indices (one in ONE mode, two in TWO mode, and three in THREE mode) each have one less bit than indicated. However, the indices are always packed into the index fields according to the table below, with the specific bit assignments of each texel following the rules just given.

Bit	Description
LSBs	texel[0][0] index
	texel[0][1] index
	texel[0][2] index
	texel[0][3] index
	texel[1][0] index
	texel[1][1] index
	texel[1][2] index
	texel[1][3] index
	texel[2][0] index
	texel[2][1] index
	texel[2][2] index
	texel[2][3] index
	texel[3][0] index
	texel[3][1] index
	texel[3][2] index
MSBs	texel[3][3] index

Endpoint Computation

The endpoints can be defined with different precision depending on mode, as shown above. This section describes how the endpoints are computed from the bits in the compression block. Each component of each endpoint follows the same steps.

If a P-bit is defined for the endpoint, it is first added as an additional LSB at the bottom of the endpoint value. The endpoint is then bit-replicated to create an 8-bit fixed point endpoint value with a range from 0x00 to 0xFF.

Palette Color Computation

The next step involves computing the color palette values that provide the available values for each texel's color. The color palette for each line consists of the two endpoint colors plus 2, 6, or 14 interpolated colors, depending on the number of bits in the indices. Again each channel is processed independently.

The equation to compute each palette color with index i, given two endpoints is as follows, using the tables below to determine the weight for each palette index:

```
palette[i] = (E0 * (64 - weight[i]) + E1 * weight[i] + 32) » 6
```

2-bit index weights:

palette index	0	1	2	3
weight	0	21	43	64

3-bit index weights:

palette index	0	1	2	3	4	5	6	7
weight	0	9	18	27	37	46	55	64

4-bit index weights:

palette index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
weight	0	4	9	13	17	21	26	30	34	38	43	47	51	55	60	64

The two end palette indices are equal to the two endpoints given that the weights are 0 and 64. In the above equation E0 and E1 represent the even-numbered and odd-numbered endpoints computed in the previous step for the component and line currently being computed.

Texel Selection

The final step is to select the appropriate palette index for each texel. This index then selects the 8-bit per channel palette value, which is interpreted as an 8-bit UNORM value for input into the filter (In BC7_UNORM_SRGB to UNORM values first go through inverse gamma conversion). This procedure differs depending on whether the mode is ONE, TWO, or THREE.

ONE Mode

In ONE mode, there is only one set of palette colors, thus there is only a single “partition set” defined, with all texels selecting line 0 and texel [0][0] being the “fix-up index” with one less bit in the index.

TWO Mode

64 partitions are defined for TWO, which are defined below. Each of the 64 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-3C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *0* and *1*) or line 1 (endpoints *2* and *3*). Each case has one texel each of "[0]" and "[1]", the index that this is at is termed the "fix-up index". These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	1	[0]	0	0	1	[0]	1	1	1	[0]	0	0	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	0	0	0	[1]	0	1	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	1	[0]	0	0	1	[0]	0	0	0
	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1
	0	0	0	1	0	1	1	1	0	1	1	1	0	0	1	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
08	[0]	0	0	0	[0]	0	1	1	[0]	0	0	0	[0]	0	0	0
	0	0	0	0	0	1	1	1	0	0	0	1	0	0	0	0
	0	0	0	1	1	1	1	1	0	1	1	1	0	0	0	1
	0	0	1	[1]	1	1	1	[1]	1	1	1	[1]	0	1	1	[1]
0C	[0]	0	0	1	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0
	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]	1	1	1	[1]
10	[0]	0	0	0	[0]	1	[1]	1	[0]	0	0	0	[0]	1	[1]	1
	1	0	0	0	0	0	0	1	0	0	0	0	0	0	1	1
	1	1	1	0	0	0	0	0	[1]	0	0	0	0	0	0	1
	1	1	1	[1]	0	0	0	0	1	1	1	0	0	0	0	0
14	[0]	0	[1]	1	[0]	0	0	0	[0]	0	0	0	[0]	1	1	1
	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1
	0	0	0	0	[1]	1	0	0	[1]	0	0	0	0	0	1	1
	0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	[1]
18	[0]	0	[1]	1	[0]	0	0	0	[0]	1	[1]	0	[0]	0	[1]	1
	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	1	[1]	0	0	0	0	1	1	0	0	1	1	0
	0	0	0	0	1	1	0	0	0	1	1	0	1	1	0	0

1C	[0]	0	0	1	[0]	0	0	0	[0]	1	[1]	1	[0]	0	[1]	1
	0	1	1	1	1	1	1	1	0	0	0	1	1	0	0	1
	[1]	1	1	0	[1]	1	1	1	1	0	0	0	1	0	0	1
	1	0	0	0	0	0	0	0	1	1	1	0	1	1	0	0
20	[0]	1	0	1	[0]	0	0	0	[0]	1	0	1	[0]	0	1	1
	0	1	0	1	1	1	1	1	1	0	[1]	0	0	0	1	1
	0	1	0	1	0	0	0	0	0	1	0	1	[1]	1	0	0
	0	1	0	[1]	1	1	1	[1]	1	0	1	0	1	1	0	0
24	[0]	0	[1]	1	[0]	1	0	1	[0]	1	1	0	[0]	1	0	1
	1	1	0	0	0	1	0	1	1	0	0	1	1	0	1	0
	0	0	1	1	[1]	0	1	0	0	1	1	0	1	0	1	0
	1	1	0	0	1	0	1	0	1	0	0	[1]	0	1	0	[1]
28	[0]	1	[1]	1	[0]	0	0	1	[0]	0	[1]	1	[0]	0	[1]	1
	0	0	1	1	0	0	1	1	0	0	1	0	1	0	1	1
	1	1	0	0	[1]	1	0	0	0	1	0	0	1	1	0	1
	1	1	1	0	1	0	0	0	1	1	0	0	1	1	0	0
2C	[0]	1	[1]	0	[0]	0	1	1	[0]	1	1	0	[0]	0	0	0
	1	0	0	1	1	1	0	0	0	1	1	0	0	1	[1]	0
	1	0	0	1	1	1	0	0	1	0	0	1	0	1	1	0
	0	1	1	0	0	0	1	[1]	1	0	0	[1]	0	0	0	0
30	[0]	1	0	0	[0]	0	[1]	0	[0]	0	0	0	[0]	0	0	0
	1	1	[1]	0	0	1	1	1	0	0	[1]	0	0	1	0	0
	0	1	0	0	0	0	1	0	0	1	1	1	[1]	1	1	0
	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
34	[0]	1	1	0	[0]	0	1	1	[0]	1	[1]	0	[0]	0	[1]	1
	1	1	0	0	0	1	1	0	0	0	1	1	1	0	0	1
	1	0	0	1	1	1	0	0	1	0	0	1	1	1	0	0
	0	0	1	[1]	1	0	0	[1]	1	1	0	0	0	1	1	0
38	[0]	1	1	0	[0]	1	1	0	[0]	1	1	1	[0]	0	0	1
	1	1	0	0	0	0	1	1	1	1	1	0	1	0	0	0
	1	1	0	0	0	0	1	1	1	0	0	0	1	1	1	0
	1	0	0	[1]	1	0	0	[1]	0	0	0	[1]	0	1	1	[1]
3C	[0]	0	0	0	[0]	0	[1]	1	[0]	0	[1]	0	[0]	1	0	0
	1	1	1	1	0	0	1	1	0	0	1	0	0	1	0	0
	0	0	1	1	1	1	1	1	1	1	1	0	0	1	1	1
	0	0	1	[1]	0	0	0	0	1	1	1	0	0	1	1	[1]

THREE Mode

64 partitions are defined for THREE, which are defined below. Each of the 64 cases shows the 4x4 block of texels, and is indexed by adding its hexadecimal row number (00-3C) to its column number (0-3). Each texel in the 4x4 block indicates whether that texel is part of line 0 (endpoints *0* and *1*), line 1 (endpoints *2* and *3*), or line 2 (endpoints *4* and *5*). Each case has one texel each of "[0]", "[1]", and "[2]", the index that this is at is termed the "fix-up index". These texels have one less bit in the index.

	0				1				2				3			
00	[0]	0	1	[1]	[0]	0	0	[1]	[0]	0	0	0	[0]	2	2	[2]
	0	0	1	1	0	0	1	1	2	0	0	1	0	0	2	2
	0	2	2	1	[2]	2	1	1	[2]	2	1	1	0	0	1	1
	2	2	2	[2]	2	2	2	1	2	2	1	[1]	0	1	1	[1]
04	[0]	0	0	0	[0]	0	1	[1]	[0]	0	2	[2]	[0]	0	1	1
	0	0	0	0	0	0	1	1	0	0	2	2	0	0	1	1
	[1]	1	2	2	0	0	2	2	1	1	1	1	[2]	2	1	1
	1	1	2	[2]	0	0	2	[2]	1	1	1	[1]	2	2	1	[1]
08	[0]	0	0	0	[0]	0	0	0	[0]	0	0	0	[0]	0	1	2
	0	0	0	0	1	1	1	1	1	1	[1]	1	0	0	[1]	2
	[1]	1	1	1	[1]	1	1	1	2	2	2	2	0	0	1	2
	2	2	2	[2]	2	2	2	[2]	2	2	2	[2]	0	0	1	[2]
0C	[0]	1	1	2	[0]	1	2	2	[0]	0	1	[1]	[0]	0	1	[1]
	0	1	[1]	2	0	[1]	2	2	0	1	1	2	2	0	0	1
	0	1	1	2	0	1	2	2	1	1	2	2	[2]	2	0	0
	0	1	1	[2]	0	1	2	[2]	1	2	2	[2]	2	2	2	0
10	[0]	0	0	[1]	[0]	1	1	[1]	[0]	0	0	0	[0]	0	2	[2]
	0	0	1	1	0	0	1	1	1	1	2	2	0	0	2	2
	0	1	1	2	[2]	0	0	1	[1]	1	2	2	0	0	2	2
	1	1	2	[2]	2	2	0	0	1	1	2	[2]	1	1	1	[1]
14	[0]	1	1	[1]	[0]	0	0	[1]	[0]	0	0	0	[0]	0	0	0
	0	1	1	1	0	0	0	1	0	0	[1]	1	1	1	0	0
	0	2	2	2	[2]	2	2	1	0	1	2	2	[2]	2	[1]	0
	0	2	2	[2]	2	2	2	1	0	1	2	[2]	2	2	1	0
18	[0]	1	2	[2]	[0]	0	1	2	[0]	1	1	0	[0]	0	0	0
	0	[1]	2	2	0	0	1	2	1	2	[2]	1	0	1	[1]	0
	0	0	1	1	[1]	1	2	2	[1]	2	2	1	1	2	[2]	1
	0	0	0	0	2	2	2	[2]	0	1	1	0	1	2	2	1
1C	[0]	0	2	2	[0]	1	1	0	[0]	0	1	1	[0]	0	0	0

	1	1	0	2	0	[1]	1	0	0	1	2	2	2	0	0	0
	[1]	1	0	2	2	0	0	2	0	1	[2]	2	[2]	2	1	1
	0	0	2	[2]	2	2	2	[2]	0	0	1	[1]	2	2	2	[1]
20	[0]	0	0	0	[0]	2	2	[2]	[0]	0	1	[1]	[0]	1	2	0
	0	0	0	2	0	0	2	2	0	0	1	2	0	[1]	2	0
	[1]	1	2	2	0	0	1	2	0	0	2	2	0	1	[2]	0
	1	2	2	[2]	0	0	1	[1]	0	2	2	[2]	0	1	2	0
24	[0]	0	0	0	[0]	1	2	0	[0]	1	2	0	[0]	0	1	1
	1	1	[1]	1	1	2	0	1	2	0	1	2	2	2	0	0
	2	2	[2]	2	[2]	0	[1]	2	[1]	[2]	0	1	1	1	[2]	2
	0	0	0	0	0	1	2	0	0	1	2	0	0	0	1	[1]
28	[0]	0	1	1	[0]	1	0	[1]	[0]	0	0	0	[0]	0	2	2
	1	1	[2]	2	0	1	0	1	0	0	0	0	1	[1]	2	2
	2	2	0	0	2	2	2	2	[2]	1	2	1	0	0	2	2
	0	0	1	[1]	2	2	2	[2]	2	1	2	[1]	1	1	2	[2]
2C	[0]	0	2	[2]	[0]	2	2	0	[0]	1	0	1	[0]	0	0	0
	0	0	1	1	1	2	[2]	1	2	2	[2]	2	2	1	2	1
	0	0	2	2	0	2	2	0	2	2	2	2	[2]	1	2	1
	0	0	1	[1]	1	2	2	[1]	0	1	0	[1]	2	1	2	[1]
30	[0]	1	0	[1]	[0]	2	2	[2]	[0]	0	0	2	[0]	0	0	0
	0	1	0	1	0	1	1	1	1	[1]	1	2	2	[1]	1	2
	0	1	0	1	0	2	2	2	0	0	0	2	2	1	1	2
	2	2	2	[2]	0	1	1	[1]	1	1	1	[2]	2	1	1	[2]
34	[0]	2	2	2	[0]	0	0	2	[0]	1	1	0	[0]	0	0	0
	0	[1]	1	1	1	1	1	2	0	[1]	1	0	0	0	0	0
	0	1	1	1	[1]	1	1	2	0	1	1	0	2	1	[1]	2
	0	2	2	[2]	0	0	0	[2]	2	2	2	[2]	2	1	1	[2]
38	[0]	1	1	0	[0]	0	2	2	[0]	0	2	2	[0]	0	0	0
	0	[1]	1	0	0	0	1	1	1	1	2	2	0	0	0	0
	2	2	2	2	0	0	[1]	1	[1]	1	2	2	0	0	0	0
	2	2	2	[2]	0	0	2	[2]	0	0	2	[2]	2	[1]	1	[2]
3C	[0]	0	0	[2]	[0]	2	2	2	[0]	1	0	[1]	[0]	1	1	[1]
	0	0	0	1	1	2	2	2	2	2	2	2	2	0	1	1
	0	0	0	2	0	2	2	2	2	2	2	2	[2]	2	0	1
	0	0	0	[1]	[1]	2	2	[2]	2	2	2	[2]	2	2	2	0

Adaptive Scalable Texture Compression (ASTC)

This section describes the data structure of the Adaptive Scalable Texture Compression (ASTC) format, as well as the decoding flow of ASTC. Also described are the header format and mipmap layout in the compressed texture file of *.astc. This is based on the reference encoder and decoder from the Khronos committee, with an extension to support multiple miplevel texture.

ASTC is a new compressed texture format with following characteristics:

1. ASTC compression format is currently only used for static texture, due to the large amount of computation and high latency required to find the optimal configuration in compression. It cannot be used to compress dynamic textures such as a shadow map.
2. ASTC is a lossy compression technique that cannot be used to compress dynamic textures which do not tolerate quality degradation.
3. ASTC has a huge range of compression ratio and block size, but these choices are fixed for each texture for all blocks at all mipmap levels.
4. ASTC has options to support compression from 1 to 4 channels for texture data.
5. ASTC can support both high and low dynamic textures.
6. ASTC can support both 2D and 3D textures.

Project	Supported Formats
CHV, BSW	2D LDR profile.

ASTC Fundamentals

This section describes some background details and new surface formats for ASTC.

Background

ASTC is a more advanced texture compression technique than the existing BC and ETC, and can reduce footprint & BW of static texture further in Graphics application by providing a texture compression solution at higher compression ratios. To best find the balance point of visual quality and compression, it provides a wide range of bit rate selection from 8bpp to 0.89 bpp in 2D, and 4.6bpp to 0.6 bpp in 3D at various block size of footprints. It also has flexibility to specify 1-4 components, selection of dual plane mode among the specified color components.

It extends the existing linear model on color distribution of each block in multiple partitions (up to 4), with flexible compact supporting on index/weight for color interpolation. ASTC also has a support of high dynamic range (HDR) image and 3D textures. The mixture of HDR and LDR data is within each block level allows a great flexibility to represent high dynamics variation at fine granularity. The support of 3D texture explores the data coherency in all 3 dimensions, without the need to mimic 3D map with 2D slices. On top of everything, void-extent regions are introduced for both 2D and 3D maps as further optimization on large constant region.

Due to the computational complexity and processing delay of the encoding process, ASTC compression encoding is always offline, and can only be used for static texture. It does not support auto mipmap generation and cannot be considered as a format for render target.

The ASTC provides a wide spectrum of bit per pixel for both 2D and 3D texture for both LDR and HDR images, hence a wide range of compression to any 2D and 3D texture.

LDR Compression Ratios:

2D Block Footprint	Bit Rate (bpp)	Compression ratio (LDR 32bpp)
4x4	8.00	4.0
5x4	6.40	5.0
5x5	5.12	6.3
6x5	4.27	7.5
6x6	3.56	9.0
8x5	3.20	10.0
8x6	2.67	12.0
10x5	2.56	12.5
10x6	2.13	15.0
8x8	2.00	16.0
10x8	1.60	20.0
10x10	1.28	25.0
12x10	1.07	29.9
12x12	0.89	36.0

HDR Compression Ratios:

Compared against fixed compression ratios of 4x or 8x on BC* formats, ASTC provides compression ratios from 4x to 36x for 2D LDR, 8x to 72x in 2D HDR maps, 7x to 54x on 3D LDR (32bpp) maps, and 14x to 108x in 3D HDR (64bpp). This can reduce bandwidth and footprint of a static 2D HDR or 3D textures to a small fractional of the existing BC formats, and greatly improve the performance on the graphic applications using these textures intensively.

Another benefit of ASTC is that, with the large range of selection of footprints and bpp, it can provide a good trade-off between quality degradation of the compressed texture and performance, due to the bandwidth and footprints reduction. This could not be achieved by any previously existing texture compression technologies.

Although ASTC has a huge benefit of bandwidth reduction, the expected performance gain in real 3D application from this technique depends on how much texture bandwidth bottleneck is relative to the throughput of computing in EU, Sampler, and other fixed function components.

New Surface Formats for ASTC Texture

The ASTC data format natively supports 14 2D block size, 10 3D block size, and each decoded format should support either UN8 (with sRGB con version) or Float16 at each color component. Following is the full list of all different surface formats as the full combination of different block shapes and UN8 or Float16 options.

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
000h	000	0	0		000 000 000	ASTC_LDR_2D_4x4_U8sRGB	8.00
008h	000	1	0		000 001 000	ASTC_LDR_2D_5x4_U8sRGB	6.40
009h	000	1	1		000 001 001	ASTC_LDR_2D_5x5_U8sRGB	5.12
011h	000	2	1		000 010 001	ASTC_LDR_2D_6x5_U8sRGB	4.27
012h	000	2	2		000 010 010	ASTC_LDR_2D_6x6_U8sRGB	3.56
021h	000	4	1		000 100 001	ASTC_LDR_2D_8x5_U8sRGB	3.20
022h	000	4	2		000 100 010	ASTC_LDR_2D_8x6_U8sRGB	2.67
031h	000	6	1		000 110 001	ASTC_LDR_2D_10x5_U8sRGB	2.56
032h	000	6	2		000 110 010	ASTC_LDR_2D_10x6_U8sRGB	2.13
024h	000	4	4		000 100 100	ASTC_LDR_2D_8x8_U8sRGB	2.00
034h	000	6	4		000 110 100	ASTC_LDR_2D_10x8_U8sRGB	1.60
036h	000	6	6		000 110 110	ASTC_LDR_2D_10x10_U8sRGB	1.28
03eh	000	7	6		000 111 110	ASTC_LDR_2D_12x10_U8sRGB	1.07
03fh	000	7	7		000 111 111	ASTC_LDR_2D_12x12_U8sRGB	0.89
040h	001	0	0		001 000 000	ASTC_LDR_2D_4x4_FLT16	8.00
048h	001	1	0		001 001	ASTC_LDR_2D_5x4_FLT16	6.40

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
					000		
049h	001	1	1		001 001 001	ASTC_LDR_2D_5x5_FLT16	5.12
051h	001	2	1		001 010 001	ASTC_LDR_2D_6x5_FLT16	4.27
052h	001	2	2		001 010 010	ASTC_LDR_2D_6x6_FLT16	3.56
061h	001	4	1		001 100 001	ASTC_LDR_2D_8x5_FLT16	3.20
062h	001	4	2		001 100 010	ASTC_LDR_2D_8x6_FLT16	2.67
071h	001	6	1		001 110 001	ASTC_LDR_2D_10x5_FLT16	2.56
072h	001	6	2		001 110 010	ASTC_LDR_2D_10x6_FLT16	2.13
064h	001	4	4		001 100 100	ASTC_LDR_2D_8x8_FLT16	2.00
074h	001	6	4		001 110 100	ASTC_LDR_2D_10x8_FLT16	1.60
076h	001	6	6		001 110 110	ASTC_LDR_2D_10x10_FLT16	1.28
07eh	001	7	6		001 111 110	ASTC_LDR_2D_12x10_FLT16	1.07
07fh	001	7	7		001 111 111	ASTC_LDR_2D_12x12_FLT16	0.89
080h	010	0	0	0	010 000 000	ASTC_LDR_3D_3x3x3_U8sRGB	4.74
090h	010	1	0	0	010 010 000	ASTC_LDR_3D_4x3x3_U8sRGB	3.56
094h	010	1	1	0	010 010 100	ASTC_LDR_3D_4x4x3_U8sRGB	2.67
095h	010	1	1	1	010 010 101	ASTC_LDR_3D_4x4x4_U8sRGB	2.00
0a5h	010	2	1	1	010 100 101	ASTC_LDR_3D_5x4x4_U8sRGB	1.60
0a9h	010	2	2	1	010 101 001	ASTC_LDR_3D_5x5x4_U8sRGB	1.28

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
0aah	010	2	2	2	010 101 010	ASTC_LDR_3D_5x5x5_U8sRGB	1.02
0bah	010	3	2	2	010 111 010	ASTC_LDR_3D_6x5x5_U8sRGB	0.85
0beh	010	3	3	2	010 111 110	ASTC_LDR_3D_6x6x5_U8sRGB	0.71
0bfh	010	3	3	3	010 111 111	ASTC_LDR_3D_6x6x6_U8sRGB	0.59
140h	101	0	0	n/a	101 000 000	ASTC_FULL_2D_4x4_FLT16	8.00
148h	101	1	0	n/a	101 001 000	ASTC_FULL_2D_5x4_FLT16	6.40
149h	101	1	1	n/a	101 001 001	ASTC_FULL_2D_5x5_FLT16	5.12
151h	101	2	1	n/a	101 010 001	ASTC_FULL_2D_6x5_FLT16	4.27
152h	101	2	2	n/a	101 010 010	ASTC_FULL_2D_6x6_FLT16	3.56
161h	101	4	1	n/a	101 100 001	ASTC_FULL_2D_8x5_FLT16	3.20
162h	101	4	2	n/a	101 100 010	ASTC_FULL_2D_8x6_FLT16	2.67
171h	101	6	1	n/a	101 110 001	ASTC_FULL_2D_10x5_FLT16	2.56
172h	101	6	2	n/a	101 110 010	ASTC_FULL_2D_10x6_FLT16	2.13
164h	101	4	4	n/a	101 100 100	ASTC_FULL_2D_8x8_FLT16	2.00
174h	101	6	4	n/a	101 110 100	ASTC_FULL_2D_10x8_FLT16	1.60
176h	101	6	6	n/a	101 110 110	ASTC_FULL_2D_10x10_FLT16	1.28
17eh	101	7	6	n/a	101 111 110	ASTC_FULL_2D_12x10_FLT16	1.07
17fh	101	7	7	n/a	101 111 111	ASTC_FULL_2D_12x12_FLT16	0.89

Value	[26] LDR/Full [25] 2D/3D [24] U8srgb /FLT16	Width 2D [23:21] 3D [23:22]	Height 2D [20:18] 3D [21:20]	Depth 2D: n/a 3D: [19:18]	Binary form	Name	(BPE)
1c0h	111	0	0	0	111 000 000	ASTC_FULL_3D_3x3x3_FLT16	4.74
1d0h	111	1	0	0	111 010 000	ASTC_FULL_3D_4x3x3_FLT16	3.56
1d4h	111	1	1	0	111 010 100	ASTC_FULL_3D_4x4x3_FLT16	2.67
1d5h	111	1	1	1	111 010 101	ASTC_FULL_3D_4x4x4_FLT16	2.00
1e5h	111	2	1	1	111 100 101	ASTC_FULL_3D_5x4x4_FLT16	1.60
1e9h	111	2	2	1	111 101 001	ASTC_FULL_3D_5x5x4_FLT16	1.28
1eah	111	2	2	2	111 101 010	ASTC_FULL_3D_5x5x5_FLT16	1.02
1fah	111	3	2	2	111 111 010	ASTC_FULL_3D_6x5x5_FLT16	0.85
1feh	111	3	3	2	111 111 110	ASTC_FULL_3D_6x6x5_FLT16	0.71
1ffh	111	3	3	3	111 111 111	ASTC_FULL_3D_6x6x6_FLT16	0.59

ASTC File Format and Memory Layout

Content for this topic is currently under development.

ASTC Header Data Structure and Amendment

The 1st block of an ASTC compression texture is a header file. Its byte layout in the original header structure in *.astc file is:

```
struct astc_header
{
    uint8_t magic[4];
    uint8_t blockdim_x;
    uint8_t blockdim_y;
    uint8_t blockdim_z;
    uint8_t xsize[3]; // x-size = xsize[0] + xsize[1] + xsize[2]
    uint8_t ysize[3]; // x-size, y-size and z-size are given in texels;
    uint8_t zsize[3]; // block count is inferred
};
```

Since there are limited ranges for block dimensions in x, y and z directions as described in following, we could store additional information in the unused upper bits of these byte fields

Block Dimension	2D	3D
blockdim_x	4, 5, 6, 8, 10, 12	3, 4, 5, 6
blockdim_y	4, 5, 6, 8, 10, 12	3, 4, 5, 6
blockdim_z	1	3, 4, 5, 6

Since blockdim_z is in the range of [1,6], only lower 3 bits of blockdim_z is used. We proposed the **Intel astc extension format** with *numLODs* stored in the upper 5 bits of the byte field used for blockdim_z. This new byte field can be defined as:

numLODs_blockdim_z = (numLODs-1) « 3 | (blockdim_z & 0x7) ;

New header:

```
struct astc_header
{
    uint8_t magic[4];
    uint8_t blockdim_x;
    uint8_t blockdim_y;
uint8_t numLODs_blockdim_z;
uint8_t xsize[3]; // width = xsize[0] + (xsize[1]«8) + (xsize[2]«16)
uint8_t ysize[3]; // height= ysize[0] + (ysize[1]«8) + (ysize[2]«16)
uint8_t zsize[3]; // depth = zsize[0] + (zsize[1]«8) + (zsize[2]«16)
// x_size, y_size and z_size are given in texels;

// block count is inferred
};
```

The driver or the software responsible for managing the memory resource will get numLODs and blockdim_z in:

numLODs = ((numLODs_blockdim_z » 3) & 0x1F) + 1;

blockdim_z = numLODs_blockdim_z & 0x7;

Data Layout in ASTC Compression File

A number of parameters are useful to determine where given pixels are located on the 2D & 3D surface. First, the width and height for each LOD level "L" is computed as:

$$W_L = \left((width \gg L) > 0\right)? width \gg L : 1)$$

$$H_L = \left((height \gg L) > 0\right)? height \gg L : 1)$$

$$D_L = \left((depth \gg L) > 0\right)? depth \gg L : 1)$$

The numbers of blocks in width, height and depth slab in each LOD are:

Nw(L) = Ceil(WL /Bw);

Nh (L) = Ceil(HL /Bh);

Ns (L) = Ceil(DL /Bd)

Where Bw, Bh and Bd is the block width, height and depth respectively.

Since ASTC has a native tile format specified by the encoding block size, the total number of blocks in each LOD level of the mipmap is described by nBL = Nw(L) * Nh (L) * Ns (L), The total number of blocks in the entire texture map is a summation of nBL's from all mipmap levels and all slabs, which are all pre-compressed via ASTC encoder. All the blocks in each LOD are in raster sequenced in width, height and then depth slab order.

Total ASTC Data Block Layout in All Mipmap Levels

The entire layout of the compression texture file looks like:

Address	Data Description
Addr0 (Base Address)	Header structure
Addr0+16	1st Data Block in LOD_0
Addr0+32	2nd Data Block in LOD_0
…	…
Addr1 = Addr0+16*nB0	Last Data Block in LOD_0
Addr1 +16	1st Data Block in LOD_1
Addr1+32	2nd Data Block in LOD_1
…	…
Addr2 = Addr1+16*nB1	Last Data Block in LOD_1
Addr2+16	1st Data Block in LOD_2
Addr2+32	2nd Data Block in LOD_2
…	…
Addr3 = Addr2+16*nB2	Last Data Block in LOD_2
…	…

Data Layout in Memory for All Mipmap Levels

The following equations for give the base address (U_offset, V_offset) in Cartesian coordinates for the starting point of each mip map at LOD L and depth slab q:

LOD=0:

U_offset (0, q) = 0;

*V_offset (0, q) = q * h0;*

LOD=1:

*U_offset (1, q) = (q%2)*w1;*

*V_offset (1, q) = D0*h0 + (q»1)*h1;*

LOD=2:

*U_offset (2, q) = (q%4)*w2;*

*V_offset (2, q) = D0*h0 + ceil(D1/2) * h1 + (q»2)*h2;*

LOD=3:

*U_offset (3, q) = (q%8)*w3;*

*V_offset (3, q) = D0*h0 + ceil(D1/2) * h1 + ceil(D2/2) * h2 + (q»3)*h3;*

••••••

Since ASTC has a native tile format specified by the encoding block size, the total number of blocks in each LOD level of the mipmap is described by nBL = Nw(L) * Nh (L) * Ns (L). The memory layout for TileY format are considered with 512bit (16Bx4) in 1 cacheline granularity, the total number of blocks is: 4*((Ceil(HL /Bh)+3)/4 * Ceil(WL /Bw) * Ceil(DL /Bd):

Here is the full list describing the total number of rows and columns of data in each mipmap for texture in ASTC format:

Table for block dimension in 2D

Block Size	ASTC Block Height (in line)	ASTC Block Width (in Byte)
4	((Ceil(HL /4) +3)/4) *4	Ceil(WL /4) * 16
5	((Ceil(HL /5) +3)/4) *4	Ceil(WL /5) * 16
6	((Ceil(HL /6) +3)/4) *4	Ceil(WL /6) * 16
8	((Ceil(HL /8) +3)/4) *4	Ceil(WL /8) * 16
10	((Ceil(HL /10) +3)/4) *4	Ceil(WL /10) * 16
12	((Ceil(HL /12) +3)/4) *4	Ceil(WL /12) * 16

Table for block dimension in 3D

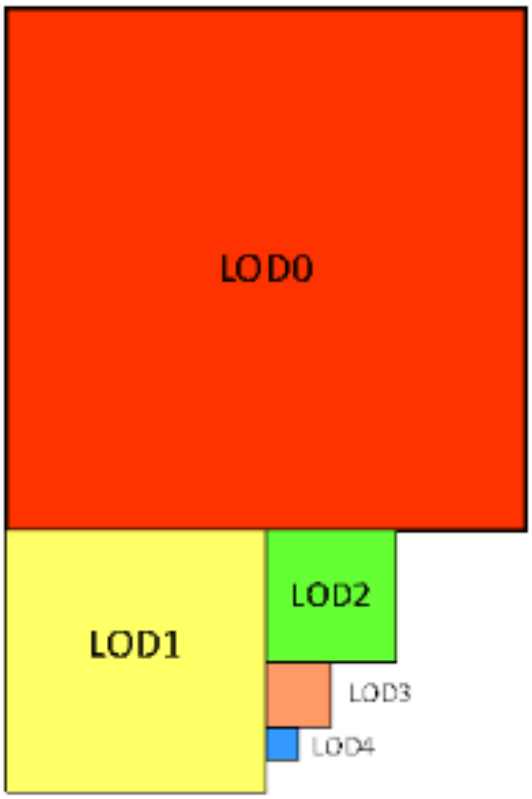
Block Size	ASTC Block Height (in line)	ASTC Block Width (in Byte)	ASTC Block Depth/slab (in slice)
3	((Ceil(HL /3) +3)/4) *4	Ceil(WL /3) * 16	Ceil(DL /3)
4	((Ceil(HL /4) +3)/4) *4	Ceil(WL /4) * 16	Ceil(DL /4)
5	((Ceil(HL /5) +3)/4) *4	Ceil(WL /5) * 16	Ceil(DL /5)
6	((Ceil(HL /6) +3)/4) *4	Ceil(WL /6) * 16	Ceil(DL /6)

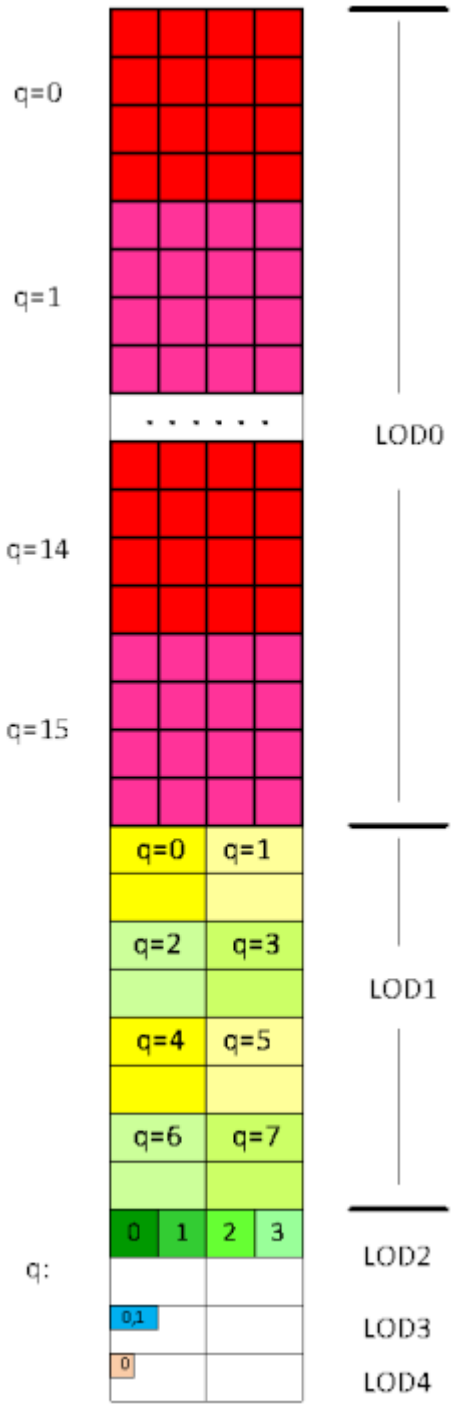
For example, an image of 64x64 with 5x5 block coding in LOD0 will have:

Block Height: (13+3)/4*4=16 (lines)

Block Width: 13 *16 = 208 (Bytes)

The following diagram illustrate the memory layout for 2D and 3D map respectively.





ASTC Data Structure

Content for this topic is currently under development.

Layout and Description of Block Data

The block data structure is described in the following table in the categories of the block being partition enabled (2-4 partitions) or disabled (only 1 partition), as well as 1 plane or dual-plane mode. Where CEM refers to Color Endpoint Mode, and CCS stands for Color Channel Selection:

Layout of Partitioning Disabled (1 partition) and Enabled (multi-partition) blocks

Partition Disable	127:19			18:17	16:13	12:11	10:0
(1 Plane)	Index Data		Color Endpoint Data		CEM	00	Index Mode
(2 Planes)	Index Data		Color Endpoint Data	CCS	CEM	00	Index Mode
Multi-partitions	127:29			28:23	22:13	12:11	10:0
(1 Plane)	Index Data	Rest of CEM	Color Endpoint Data	CEM (Initial 6 bits)	Partition Index	Part	Index Mode
(2 Planes)	Index Data	CCS	Rest of CEM	Color Endpoint Data	CEM (Initial 6 bits)	Partition Index	Part	Index Mode

The 11 bit “Index mode” field specifies how the Texel Index Data is encoded. The bit encoding of this field is listed in next two tables, one for the 2D and one for the 3D.

The “Part” field specifies the number of partitions minus one. If dual plane mode is enabled, the number of partitions must be 3 or fewer. In case 4 partitions in such situation are specified, the error value is returned for all texels in the block. The size and layout of the extra configuration data depends on the number of partition, and the number of planes in the image.

Partitioning

For any non-void extend region, each block is subdivided into 1, 2, 3 or 4 partitions, with a separate color endpoint pair for each partition. The number of partitions is specified by the partition count-1 in bits [12:11] of block data. If 2 or more partitions are selected, partitioning is enabled, the 10 bit partition index is then used to select one from 1024 partitioning patterns, where the total set of patterns supported in ASTC depends on the partition count and block size. The partitioning patterns are produced generatively, which supports a very large set of partitioning patterns for different block sizes with a modest number of hardware gates implementation.

Index Mode

The “Index mode” field specifies how the Texel Index Data is encoded. The bit encoding of this field is listed in next two tables, one for the 2D and one for the 3D.

The Index Mode field specifies the width (N), height (M) and depth (Q) of the grid of indices, what range of values they use, and whether dual index planes are present. The index ranges are encoded using a 3 bit value R, which is interpreted together with a precision bit H, as follows:

Mode R(r2 r1 r0)	Low-precision (H=0)					High-precision (H=1)			
	Index Range	Trits	Quints	Bits		Index Range	Trits	Quints	Bits
0 0 0	Invalid					Invalid			
0 0 1	Invalid					Invalid			
0 1 0	[0,1]			1		[0,9]		1	1
0 1 1	[0,2]	1				[0,11]	1		2
1 0 0	[0,3]			2		[0,15]			4
1 0 1	[0,4]		1			[0,19]		1	2
1 1 0	[0,5]	1		1		[0,23]	1		3
1 1 1	[0,7]			3		[0,31]			5

Each index value is encoded using the specified number of Trits, Quints and Bits. The details of this encoding can be found in Section - *Integer Sequence Encoding*. Due to the encoding of the R field, bits r2 and r1 cannot both be zero

The number of indices provided for a block is not tied to the block size in any way, instead, the indices form an N*M*Q ordered grid. N, M and Q are specified on a per-block basis rather then being a global texture property. For 2D blocks, N and M can be set to any value from 2 to 12 while Q is fixed at 1; for 3D blocks, N, M and Q can be set to any value from 2 to 5. The range used for each index can be set separately for each block. The Index Bit Mode field species the values of N, M, Q and the range; it also specifies whether Dual Index Planes are present or not as well.

The D bit in following tables is set to indicate dual-plane mode. In this mode, the maximum allowed number of partitions is 3. The size of the grid in each dimension must be less than or equal to the corresponding dimension of the block footprint. If the grid size is greater than the footprint dimension in any axis, then this is an illegal block encoding and all texels will decode to the error color.

For 2D blocks, the index mode field is laid out as follows:

The bit encoding of the index mode field for 2D Blocks

Bits									Width	Height	Notes
10	9	8	7	6	5	4	3:2	1:0	N	M	
D	H	B		A		r0	0 0	r2 r1	B+4	A+2	
D	H	B		A		r0	0 1	r2 r1	B+8	A+2	
D	H	B		A		r0	1 0	r2 r1	A+2	B+8	
D	H	0	B	A		r0	1 1	r2 r1	A+2	B+6	
D	H	1	B	A		r0	1 1	r2 r1	B+2	A+2	
D	H	0	0	A		r0	r2 r1	0 0	12	A+2	
D	H	0	1	A		r0	r2 r1	0 0	A+2	12	
D	H	1	1	0	0	r0	r2 r1	0 0	6	10	
D	H	1	1	0	1	r0	r2 r1	0 0	10	6	
B		1	0	A		r0	r2 r1	0 0	A+6	B+6	D=0, H=0
x	x	1	1	1	1	1	1 1	0 0	-	-	Void-Extent
x	x	1	1	1	x	x	x x	0 0	-	-	Reserve
x	x	x	x	x	x	x	0 0	0 0	-	-	Reserve

Note that, due to the encoding of the R field (r0, r1, r2), bits r2 and r1 cannot both be zero, which disambiguates the first five rows from the rest of the table. The penultimate row of the table is reserved only if bits [5:2] are not all 1, in which case it encodes a void-extent block (as shown in the previous row)

For 3D blocks, the index mode field is laid out as follows:

3D Index Mode Layout

Bits											Notes			
10	9	8	7	6	5	4	3	2	1	0	N	M	Q	(D, H)
D	H	B		A		r0	C		r2	r1	A+2	B+2	C+2	
B		0	0	A		r0	r2	r1	0	0	6	B+2	A+2	(0, 0)
B		0	1	A		r0	r2	r1	0	0	A+2	6	B+2	(0, 0)
B		1	0	A		r0	r2	r1	0	0	A+2	B+2	6	(0, 0)
D	H	1	1	0	0	r0	r2	r1	0	0	6	2	2	
D	H	1	1	0	1	r0	r2	r1	0	0	2	6	2	
D	H	1	1	1	0	r0	r2	r1	0	0	2	2	6	
x	x	1	1	1	1	1	1	1	0	0	-	-	-	Void-Extent
x	x	1	1	1	1	x	x	x	0	0	-	-	-	Reserve
x	x	x	x	x	x	x	0	0	0	0	-	-	-	Reserve

The D bit is set to indicate dual-plane mode:

1: dual index planes are used

0: single index plane is used

In this mode, the maximum allowed number of partitions is 3. The size of the grid in each dimension must be less than or equal to the corresponding dimension of the block footprint. If the grid size is greater than the footprint dimension in any axis, then this is an illegal block encoding and all texels will decode to the error color. The penultimate row of the table is reserved only if bits [4:2] are not all 1, in which case it encodes a void-extent block (as shown in the previous row).

H: Index Range Bit:

1: the High-Precision group is selected.

0: The Low-Precision group is selected.

Here is the detail description:

- The encoding of xx111111100 is for the void-extent block.
- The pattern xxxxxxx0000 (the bottom 4 bits being 0000b) is reserved for future extension, and should result a NaN-vector when such a pattern is decoded.
- Any encodings not listed in the table are considered invalid and result in undened behavior if encountered by decoders.

Given the limitation of the fix length of 128 bits per block, there are restrictions that will not allow every possible encoding:

- The total number of indexes (N*M*Q for single index plane, 2*N*M*Q for dual index planes) must not exceed 64.
- The length of the Index Integer Sequence must not exceed 96 bits.
- The length of the Index Integer Sequence must be at least 24 bits.
- The above restriction, combined with the other field widths of the format, implicitly restricts the Color Integer Sequence to a maximum of 75 bits.
- Blocks that violate these restrictions are not legally produced by the encoder, result a vector of NaNs if encountered by decoders.

Here is how the indices in each block are encoded and stored:

- They are encoded using the Integer Sequence Encoding method described in Appendix.
- The resulting bit-sequence is then bit-reversed, and stored from the top of the block downwards. The ordering of the indices in the Integer Sequence is a simple scan line-like ordering.

The indices are used in two steps to interpolate between two endpoint colors for each texel.

- First, they are scaled from whatever interval they were to the range [0,64];
- The resulting value is then used as a weight to interpolate between the two endpoints.

Index Planes

Depending on the Index Bits mode selected, an ASTC compressed block may offer 1 or 2 index planes. In the case of 2 index planes, two indices rather than just one are supplied for each texel that receives indices. Of these two indices, the first one is used for a weighted sum of three of the color components; the second is used for a weighted sum of the fourth color component. If only 1 index plane is present, it applies to all four color components.

If two index planes are used, then a 2-bit bit field is needed to indicate which of the color components the second index plane applies to. These two bits are stored just below the index bits, except in the case where leftover color endpoint type bits are present; in that case, these two bits are stored just below the leftover color endpoint type bits. This two-bit bit-field has the following layout:

Channel	Red	Green	Blue	Alpha
Value	0	1	2	3

If index infill is present while two index planes are being used, then index infill is performed on each index plane separately. If two index planes are used, the indexes are stored interleaved: the first index belongs to the first index plane, the second index belongs to the second index plane, the third index belongs to the first index plane, and so on.

Index Infill Procedure

In ASTC, each block has an N*M*Q ordered grid of indices. N, M and Q may or may not match the dimensions of the actual block (e.g. it is possible to encode a 5x3 grid for an 8x8 block); if they don't match, then the grid is scaled so that its corner indexes align with the corner texels of the block, a bilinear index infill procedure is defined to interpolate an index for each texel. This procedure picks 1 to 4 indexes, and assigns each of them a weight; these weights are always a multiple of 1/16. The exact details of this interpolation procedure are specified below.

Color Endpoint Mode

In single-partition mode, the Color Endpoint Mode (CEM) field stores one of 16 possible values. Each of these specifies how many raw data values are encoded, and how to convert these raw values into two RGBA color endpoints. They can be summarized as follows:

List of Color Endpoint Modes

CEM	Description	Class	# of integers to represent each pair of color end points
0	LDR Luminance or Alpha, direct	0	2
1	LDR Luminance, base+offset	0	2
2	HDR Luminance, large range	0	2
3	HDR Luminance, small range	0	2
4	LDR Luminance+Alpha, direct	1	4
5	LDR Luminance+Alpha, base+offset	1	4
6	LDR RGB, base+scale	1	4
7	HDR RGB, base+scale	1	4
8	LDR RGB, direct	2	6
9	LDR RGB, base+offset	2	6
10	LDR RGB, base+scale plus two A	2	6
11	HDR RGB, direct	2	6
12	LDR RGBA, direct	3	8: D=0; 6: D=1
13	LDR RGBA, base+offset	3	8: D=0; 6: D=1
14	HDR RGB, direct + LDR Alpha	3	8: D=0; 6: D=1
15	HDR RGB, direct + HDR Alpha	3	8: D=0; 6: D=1

Project	Description
CHV, BSW	LDR modes are supported in ASTC LDR profile, which is enabled since CHV, BSW.

In 2-4 partition modes, the encoding of Color Endpoint Modes are listed in following tables, where the endpoint mode representation may take from 6 to 14 bits, of which the first 6 bits are stored just after the partition indices, and the remaining bits are stored just below the index bits at variable position in the remaining space.

Partition / Class Types		High bits				[1:0]
Same Class	**6b [5:0]**	[5:2] Color Endpoint Mode				0 0
Different Classes	**2-Partions 8b [7:0]**	[7:6] Mode in P1	[5:4] Mode in P0	[3:3] Class Select for P1	[2:2] Class Select for P0	0 1 (Class 0 & 1)
	3-Partions 11b [10:0]			[10:9] Mode in P2	[8:7] Mode in P1	1 0 (Class 1 & 2)
		[6:5] Mode in P0	[4:4] Class Select for P2	[3:3] Class Select for P1	[2:2] Class Select for P0	
	4-Partions 14b [13:0]	[13:12] Mode in P2	[11:10] Mode in P1	[9:8] Mode in P2	[7:6] Mode in P1	1 1 (Class 2 & 3)
		[5:5] Class Select for P3	[4:4] Class Select for P2	[3:3] Class Select for P1	[2:2] Class Select for P0	

More specifically, if the CEM selector value in bits [24:23] is not 00, then data layout is as follows:

List of Color Endpoint Class Types encoding under multi-partitions

Partitions							...	28	27	26	25	24	23
2	...	Index	M1				...	M0		C1	C0	CEM	
3	...	Index	M2	M1	M0		...	M0	C2	C1	C0	CEM	
4	...	Index	M3	M2	M1	M0	...	C3	C2	C1	C0	CEM	

In this view, each partition *i* has two fields. C*i* is the class selector bit, choosing between the two possible CEM classes (0 indicates the lower of the two classes), and M*i* is a two-bit field specifying the low bits of the color endpoint mode within that class. The additional bits appear at a variable bit position, immediately below the texel index data. The ranges used for the data values are not explicitly specified. Instead, they are derived from the number of available bits remaining after the configuration data and index data have been specified. Details of the decoding procedure for Color Endpoints can be found later.

Color Endpoint Data Size Determination

The size of the data used to represent color endpoints is not explicitly specified. Instead, it is determined from the index mode and number of partitions as follows:

```
config_bits = 17;
if (num_partitions>1)
if (single_CEM)
config_bits = 29;
else
config_bits = 24 + 3*num_partitions;

num_indices = M * N * Q; // size of index grid

if (dual_plane)
config_bits += 2;
num_indices *= 2;

index_bits = floor(num_indices*8*trits_in_index_range/5) +
floor(num_indices*7*quints_in_index_range/3) +
num_indices*bits_in_index_range;

remaining_bits = 128 - config_bits - index_bits;
num_CEM_pairs = base_CEM_class+1 + count_bits(extra_CEM_bits);
```

The CEM value range is then looked up from a table indexed by remaining bits and num_CEM_pairs. This table is initialized such that the range is as large as possible, consistent with the constraint that the number of bits required to encode num_CEM_pairs pairs of values is not more than the number of remaining bits. An equivalent iterative algorithm would be:

```
num_CEM_values = num_CEM_pairs*2;
for(range = each possible CEM range in descending order of size)
{
   CEM_bits = floor(num_CEM_values*8*trits_in_CEM_range/5) +
      floor(num_CEM_values*7*quints_in_CEM_range/3) +
      num_CEM_values*bits_in_CEM_range;
if (CEM_bits <= remaining_bits)
   break;
}
return range;
```

In cases where this procedure results in unallocated bits, these bits are not read by the decoding process and can have any value.

Void-Extent Blocks

As noted in the index mode, a specifically type of encoding is the void-extended type (2D), an efficient way to encode a constant color for large blocks of regions in texture. The data structure of a void extent is listed in following 2 tables as 2D and 3D blocks respectively.

Layout of 2D Void-Extend Block, being supported in LDR since CHV, BSW.

127:112	111:96	95:80	79:64	63:51	50:38	37:25	24:12	11:10	9	8:0
A	B	G	R	T_high	T_low	S_high	S_low	Res:11	H	111111100

Bit 9 H is the Dynamic Range flag, which indicates the format in which colors are stored. A 0 value indicates LDR, in which case the color components are stored as UNORM16 values. A 1 indicates HDR, in which case the color components are stored as FP16 values. If a void-extent block with HDR values is decoded in LDR mode, then the result will be the error color, opaque magenta, for all texels within the block. The low and height coordinate values are treated as unsigned integers and then normalized into the range 0..1 (by dividing by 2^{13}-1 for 2D or 2^{9}-1, for 3D respectively). The high values for each dimension must be greater than the corresponding low values, unless they are all all-1s. If all the coordinates are all-1s, then the void extent is ignored, and the block is simply a constant color block.The existence of single-color blocks with void extents must not produce results different from those obtained if these single-color blocks are defined without void-extents. Any situation in which the results would differ is invalid. Results from invalid void extents are undefined. If a void-extent appears in a MIPmap level other than the most detailed one, then the extent will apply to all of the more detailed levels too. This allows decoders to avoid sampling more detailed MIPmaps. If the more detailed MIPmap level is *not* a constant color in this region, then the block may be marked as constant color, but without a void extent, as detailed above. If a void-extent extends to the edge of a texture, then filtered texture colors may not be the same color as that specified in the block, due to texture border colors,

wrapping, or cube face wrapping. Care must be taken when updating or extracting partial image data that void-extents in the image do not become invalid.

Decoding Process

Overview Decoding Flow

The goal for this feature is to reconstruct a cacheline (512b) of a target texture data at 4x4 region in UNORM8 A8R8G8B8 or 4x2 in FLT16 A16R16G16B16 with certain performance target, given the input texture coordinate (s,t,r). The scope of the u-architecture includes

- The additional surface format of the post decoding block, and the footprint (equivalent bpp). These are both global to each texture surface, and can be passed to the Sampler in the surface state via sampler messages.
- With post-scaled texture coordinate (u, v, p), the additional address calculation in FT to find the particular block location relative to the native block size specified in the surface state, as well as the relative texel position within that block. Assuming the block size for the block is Bu, Bv, Bp, the dimensions of a 2D surface as measured in block size tsize is:

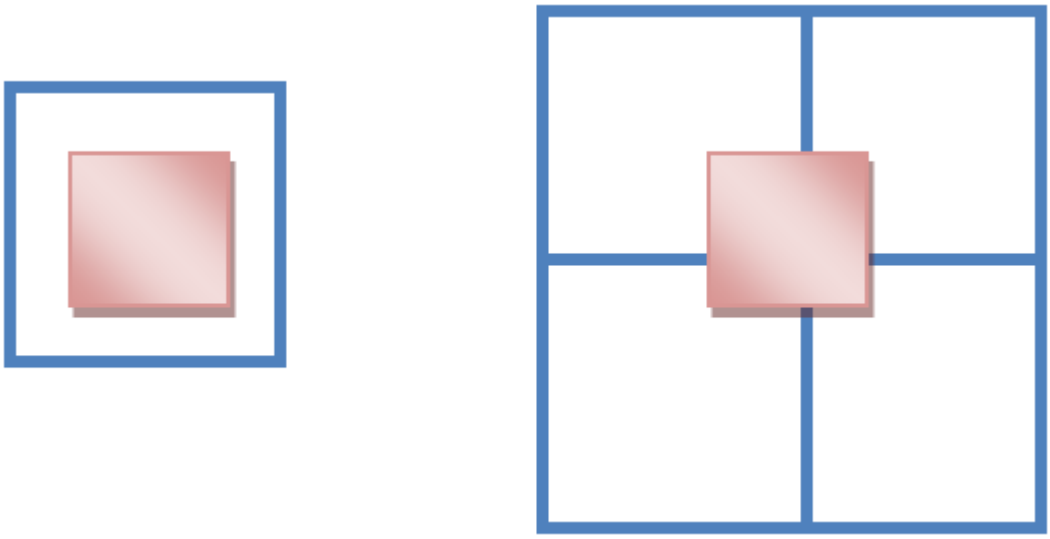
```
bw = MAX ( 2, (w+ tsize -1)/ tsize )
bh  = MAX (2, (h+ tsize -1)/ tsize)
```

Here the division is an integer division. The relationship between non-negative image coordinates [row,col] =[u, v] and block coordinates is

```
bu  = u / tsize  ;  buu = u % tsize;
bv  = v / tsize  ;  bvv =  v % tsize;
bp  = p / tsize  ;  bpp = p % tsize;
```

- With the selected sets of block size from 4x4 to 12x12 in 2D and 3x3x3 to 6x6x6 in 3D maps,1~4 blocks of source texture needs to be fetched, depending on whether the destination tile size (4x2 in FLT16 or 4x4 in UNORM8888) is inclusive or come across a few source blocks, as shown in Fig.

Destination tile is inclusive within one tile or across up to 4 tiles in source texture region



- Decode 1 to 4 128-bit ASTC compressed blocks fetched from DRAM in Sampler from ASTC compression format to either UNORM8 (LDR) or FLT16(HDR), reconstruct the texels needed in the texture filtering stage. The total decoding processing include:

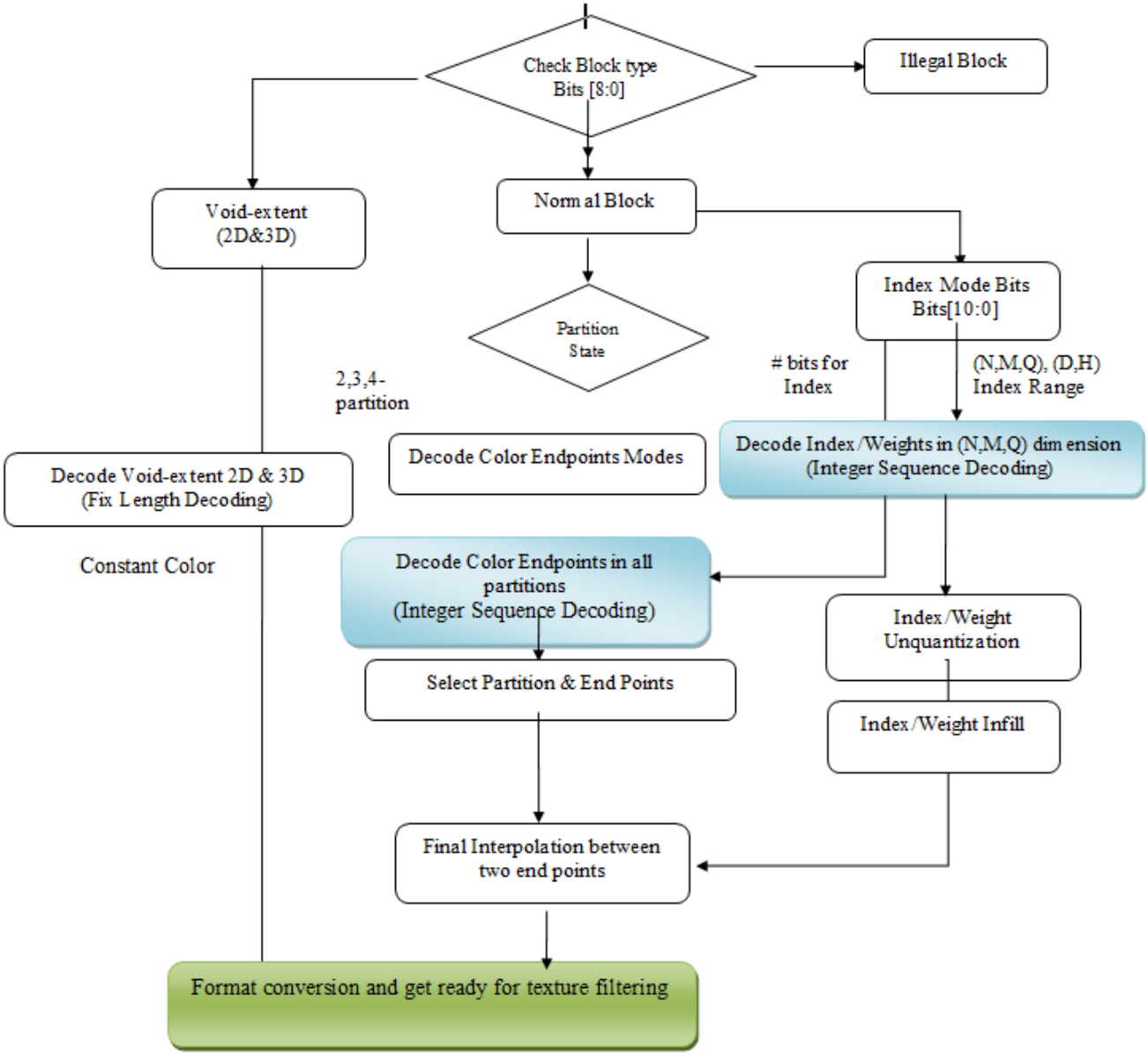
Front End Decoding Processing:

1. Detect if an ASTC block is a void-extent type, illegal type, or a normal non-void-extent type.
2. Decode the partition state – number of partitions in the current block.
3. Decode the index mode for the block include the partition seed and (N,M,Q) dimension of the compact sampling domain.
4. Decode the color endpoints modes in each partition.
5. Calculate the bit position and total # of bits used for Index.
6. Calculate the bit position and total # of bits and # of integers used in the Color endpoints in all partitions within the block.
7. With Integer Sequence Decoding, get all the indices in the compact domain defined by NxMxQ grid.
8. With Integer Sequence Decoding, get all the color end points from 16 modes in FLT16 for all partitions.

Back End Decoding Processing:

1. Reconstruct the indices at the selected sampling locations with infill scaling.
2. Find the partition from the partition seed at each sampling location.
3. Reconstruct the texture color value with the index and the pair of color end points at each sampling location.
4. If Block type is void extent, get the constant color from the high 64 bits and assign to the sampling location.
5. Convert the data to UNORM8 if LDR data is needed for the subsequent FL filtering process. Under void-extent block type

Following is the flow diagram of the decoding process:



Integer Sequence Encoding

Both the index data and the endpoint color data are variable width, and are specified using a sequence of integer values. The range of each value in a sequence (e.g. a color index) is constrained. Since it is often the case that the most efficient range for these values is not a power of two, each value sequence is encoded using a technique known as "integer sequence encoding". This allows efficient, hardware-friendly packing and unpacking of values with non-power-of-two ranges. In a sequence, each value has an identical range. The range is specified in one of the following forms:

Value range	MSB encoding	LSB encoding	Value	Block	Packed block size
0 .. 2^n-1	-	*n* bit value *m* (*n* <= 8)	*m*	1	*n*
0 .. (3 * 2^n)-1	Base-3 "trit" value *t*	*n* bit value *m* (*n* <= 6)	*t* * 2^n + *m*	5	8 + 5*n
0 .. (5 * 2^n)-1	Base-5 "quint" value *q*	*n* bit value *m* (*n* <= 5)	*q* * 2^n + *m*	3	7 + 3*n

Since 3^5 is 243, it is possible to pack five trits into 8 bits (which has 256 possible values), so a trit can effectively be encoded as 1.6 bits. Similarly, since 5^3 is 125, it is possible to pack three quints into 7 bits (which has 128 possible values), so a quint can be encoded as 2.33 bits.

The encoding scheme packs the trits or quints, and then interleaves the *n* additional bits in positions that satisfy the requirements of an arbitrary length stream. This makes it possible to correctly specify lists of values whose length is not an integer multiple of 3 or 5 values. It also makes it possible to easily select a value at random within the stream. If there are insufficient bits in the stream to fill the final block, then unused (higher order) bits are assumed to be 0 when decoding.

To decode the bits for value number *i* in a sequence of bits *b*, both indexed from 0, perform the following:

If the range is encoded as *n* bits per value, then the value is bits *b*[*i***n*+*n*-1:*i***n*] – a simple multiplexing operation.

If the range is encoded using a trit, then each block contains 5 values (v0 to v4), each of which contains a trit (t0 to t4) and a corresponding LSB value (m0 to m4). The first bit of the packed block is bit floor(*i*/5)*(8+5**n*). The bits in the block are packed as follows (in this example, *n* is 4):

Trit-based Packing

27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T7			m4		T6		T5		m3		T4		m2		T3	T2		m1				T1	T0	m0			

The five trits t0 to t4 are obtained by bit manipulations of the 8 bits T[7:0] as follows:

```
if T[4:2] = 111
   C = { T[7:5], T[1:0] }; t4 = t3 = 2
else
   C = T[4:0]
   if T[6:5] = 11
      t4 = 2; t3 = T[7]
   else
      t4 = T[7]; t3 = T[6:5]
```

```
if C[1:0] = 11
   t2 = 2; t1 = C[4]; t0 = { C[3], C[2]&~C[3] }
else if C[3:2] = 11
   t2 = 2; t1 = 2; t0 = C[1:0]
else
   t2 = C[4]; t1 = C[3:2]; t0 = { C[1], C[0]&~C[1] }
```

Endpoint Unquantization

Each color endpoint is specified as a sequence of integers in a given range. These values are packed using integer sequence encoding, as a stream of bits stored from just above the configuration data, and growing upwards. Once unpacked, the values must be unquantized from their storage range, returning them to a standard range of 0..255. For bit-only representations, this is simple bit replication from the most significant bit of the value. For trit or quint-based representations, this involves a set of bit manipulations and adjustments to avoid the expense of full-width multipliers. This procedure ensures correct scaling, but scrambles the order of the decoded values relative to the encoded values. This must be compensated for using a table in the encoder.

The initial inputs to the procedure are denoted A, B, C and D and are decoded using the range as follows:

Range	Trits	Quints	Bits	Bit value	A (9 bits)	B (9 bits)	C (9 bits)	D (3 bits)
0..5	1		1	a	aaaaaaaaa	000000000	204	Trit value
0..9		1	1	a	aaaaaaaaa	000000000	113	Quint value
0..11	1		2	ba	aaaaaaaaa	b000b0bb0	93	Trit value
0..19		1	2	ba	aaaaaaaaa	b0000bb00	54	Quint value
0..23	1		3	cba	aaaaaaaaa	cb000cbcb	44	Trit value
0..39		1	3	cba	aaaaaaaaa	cb0000cbc	26	Quint value
0..47	1		4	dcba	aaaaaaaaa	dcb000dcb	22	Trit value
0..79		1	4	dcba	aaaaaaaaa	dcb0000dc	13	Quint value
0..95	1		5	edcba	aaaaaaaaa	edcb000ed	11	Trit value
0..159		1	5	edcba	aaaaaaaaa	edcb0000e	6	Quint value
0..191	1		6	fedcba	aaaaaaaaa	fedcb000f	5	Trit value

These are then processed as follows:

T= D * C + B;

T = T ^ A;

T = (A & 0x80) | (T » 2);

The multiply in the first line is nearly trivial as it only needs to multiply by 0, 1, 2, 3 or 4.

LDR Endpoint Decoding

The decoding method used depends on the Color Endpoint Mode (CEM) field, which specifies how many values are used to represent the endpoint. The CEM field also specifies how to take the *n* unquantized color endpoint values v0 to vn-1 and convert them into two RGBA color endpoints e0 and e1. The HDR Modes are more complex and do not fit neatly into the table. They are documented in following section. The LDR methods can be summarized as follows.

Color Endpoint Modes

CEM	Range	Description	# of end points	Endpoints Reconstruction
0	LDR	Luminance, direct	2	e0=(v0,v0,v0,0xFF); e1=(v1,v1,v1,0xFF);

1	LDR	Luminance
		base+offset
			2	L0 = (v0»2)|(v1&0xC0); L1=L0+(v1&0x3F);
				if (L1>0xFF) { L1=0xFF; }
				e0=(L0,L0,L0,0xFF); e1=(L1,L1,L1,0xFF);

2	HDR	Luminance, large range	2	See next Section
3	HDR	Luminance, small range	2	See next Section
4	LDR	Luminance+Alpha Direct	4	e0=(v0,v0,v0,v2); e1=(v1,v1,v1,v3);

5	LDR	Luminance+Alpha, base+offset	4	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				e0=(v0,v0,v0,v2); e1=(v0+v1,v0+v1,v0+v1,v2+v3);
				clamp_unorm8(e0); clamp_unorm8(e1);

6	LDR	RGB base+scale	4	e0=(v0*v3»8,v1*v3»8,v2*v3»8, 0xFF); e1=(v0,v1,v2,0xFF);
7	HDR	RGB, base+scale	4	See next Section

8	LDR	RGB
		Direct
			6	s0= v0+v2+v4; s1= v1+v3+v5;
				if (s1>=s0){e0=(v0,v2,v4,0xFF); e1=(v1,v3,v5,0xFF); }
				else { e0=blue_contract(v1,v3,v5,0xFF);
				e1=blue_contract(v0,v2,v4,0xFF); }

9	LDR	RGB
		base+offset
			6	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				bit_transfer_signed(v5,v4);

CEM	Range	Description	# of end points	Endpoints Reconstruction

				if(v1+v3+v5 >= 0)
				{ e0=(v0,v2,v4,0xFF); e1=(v0+v1,v2+v3,v4+v5,0xFF); }
				else
				{ e0=blue_contract(v0+v1,v2+v3,v4+v5,0xFF);
				e1=blue_contract(v0,v2,v4,0xFF); }
				clamp_unorm8(e0); clamp_unorm8(e1);

10	LDR	RGB base+scale plus two A	6	e0=(v0*v3»8,v1*v3»8,v2*v3»8, v4); e1=(v0,v1,v2, v5);
11	HDR	RGB	6	See next Section

12	LDR	RGBA
		direct
			8	s0= v0+v2+v4; s1= v1+v3+v5;
				if (s1>=s0){e0=(v0,v2,v4,v6); e1=(v1,v3,v5,v7); }
				else { e0=blue_contract(v1,v3,v5,v7);
				e1=blue_contract(v0,v2,v4,v6); }

13	LDR	RGBA
		base+offset
			8	bit_transfer_signed(v1,v0);
				bit_transfer_signed(v3,v2);
				bit_transfer_signed(v5,v4);
				bit_transfer_signed(v7,v6);
				if(v1+v3+v5>=0) { e0=(v0,v2,v4,v6);
				e1=(v0+v1,v2+v3,v4+v5,v6+v7); }
				else { e0=blue_contract(v0+v1,v2+v3,v4+v5,v6+v7);
				e1=blue_contract(v0,v2,v4,v6); }
				clamp_unorm8(e0); clamp_unorm8(e1);

14	HDR	RGB + LDR Alpha	8	See next Section
15	HDR	RGB + HDR Alpha	8	See next Section

Mode 14 is special in that the alpha values are interpolated linearly, but the color components are interpolated logarithmically. This is the only endpoint format with mixed-mode operation, and will return the error value if encountered in LDR mode. The bit_transfer_signed procedure transfers a bit from one signed byte value (a) to another (b). The result is an 8-bit signed integer value and a 6-bit integer value sign extended to 8 bits. Note that, as is often the case, this is easier to express in hardware than in C:

```
bit_transfer_signed(uint16_t& a, uint16_t& b)
{
b »= 1;
b |= a & 0x80;
a »= 1;
a &= 0x3F;
if( (a&0x20)!=0 ) a-=0x40;
}
```

For the purposes of this pseudocode, the signed bytes are passed in as unsigned 16-bit integers because the semantics of a right shift on a signed value in C are undefined.

The blue_contract procedure is used to give additional precision to RGB colors near grey:

```
color blue_contract( int r, int g, int b, int a )
{
color c;
c.r = (r+b) » 1;
c.g = (g+b) » 1;
c.b = b;
c.a = a;
return c;
}
```

The clamp_unorm8 procedure is used to clamp a color into the UNORM8 range:

```
void clamp_unorm8(color c)
{
if(c.r < 0) {c.r=0;} else if(c.r > 255) {c.r=255;}
if(c.g < 0) {c.g=0;} else if(c.g > 255) {c.g=255;}
if(c.b < 0) {c.b=0;} else if(c.b > 255) {c.b=255;}
```

if(c.a < 0) {c.a=0;} else if(c.a > 255) {c.a=255;}

}

HDR Endpoint Decoding

The 6 HDR CEM modes on color endpoints reconstruction and surface formats are only used in full-profile ASTC texture in float 16 bit.

- HDR Endpoint Mode 2: HDR Luminance, large range
- HDR Endpoint Mode 3: HDR Luminance, small range
- HDR Endpoint Mode 7: HDR RGB, base + scale
- HDR Endpoint Mode 11: HDR RGB, direct
- HDR Endpoint Mode 14: HDR RGB, direct + LDR Alpha
- HDR Endpoint Mode 15: HDR RGB, direct + HDR Alpha

HDR Endpoint Mode 2 (HDR Luminance, Large Range)

Mode 2 represents luminance-only data with a large range. It encodes using two values (v0, v1). The complete decoding procedure is as follows:

```
If (v1 >= v0)
{
   y0 = (v0 « 4);
   y1 = (v1 « 4);
}
else {
   y0 = (v1 « 4) + 8;
   y1 = (v0 « 4) - 8;
}
// Construct RGBA result (0x780 is 1.0f)

e0 = (y0, y0, y0, 0x780);
e1 = (y1, y1, y1, 0x780);
```

HDR Endpoint Mode 3 (HDR Luminance, Small Range)

Mode 3 represents luminance-only data with a small range. It packs the bits for a base luminance value, together with an offset, into two values (v0, v1):

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	M	L[6:0]						
V1	X[3:0]				D[3:0]			

The bit field marked as X allocates different bits to L or d depending on the value of the mode bit M. The complete decoding procedure is as follows:

```
// Check mode bit and extract.
    If ((v0&0x80) !=0)
{
y0 = ((v1 & 0xE0) « 4) | ((v0 & 0x7F) « 2);
d = (v1 & 0x1F) « 2;
}
else {
y0 = ((v1 & 0xF0) « 4) | ((v0 & 0x7F) « 1);
d = (v1 & 0x0F) « 1;
}
// Add delta and clamp
y1 = y0 + d;
if(y1 > 0xFFF) { y1 = 0xFFF; }
// Construct RGBA result (0x780 is 1.0f)
e0 = (y0, y0, y0, 0x780);
e1 = (y1, y1, y1, 0x780);
```

HDR Endpoint Mode 7 (HDR RGB, Base+Scale)

Mode 7 packs the bits for a base RGB value, a scale factor, and some mode bits into the four values (v0, v1, v2, v3).

HDR Mode 7 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	M[3]	M[2]	R[5:0]					
V1	M[1]	X0	X1	G[4:0]				
V2	M[0]	X2	X3	B[4:0]				
V3	X4	X5	X6	S[4:0]				

The mode bits M[0:3] are a packed representation of an endpoint bit mode, together with the major component index. For modes 0 to 4, the component (red, green, or blue) with the largest magnitude is identified, and the values are swizzled to ensure that it is decoded from the red channel. The endpoint bit mode is used to determine the number of bits assigned to each component of the endpoint, and the destination of each of the extra bits X0 to X6, as follows:

Endpoint Bit Mode

	Number of bits					Description of Extra Bits						
Mode	R	G	B	Scale		X0	X1	X2	X3	X4	X5	X6
0	11	5	5	7		R[9]	R[8]	R[7]	R[10]	R[6]	S[6]	S[5]
1	11	6	6	5		R[8]	G[5]	R[7]	B[5]	R[6]	R[10]	R[9]
2	10	5	5	8		R[9]	R[8]	R[7]	R[6]	S[7]	S[6]	S[5]
3	9	6	6	7		R[8]	G[5]	R[7]	B[5]	R[6]	S[6]	S[5]
4	8	7	7	6		G[6]	G[5]	B[6]	B[5]	R[6]	R[7]	S[5]
5	7	7	7	7		G[6]	G[5]	B[6]	B[5]	R[6]	S[6]	S[5]

The complete decoding procedure is as follows:

```
// Extract mode bits and unpack to major component and mode.
int modeval = ((v0 & 0xC0) » 6) | ((v1 & 0x80) » 5) | ((v2 & 0x80) » 4);
int majcomp;
int mode;
if( (modeval & 0xC ) != 0xC ) { majcomp = modeval » 2; mode = modeval & 3; }
else if( modeval != 0xF ) { majcomp = modeval & 3; mode = 4; }
else { majcomp = 0; mode = 5; }
// Extract low-order bits of r, g, b, and s.
int red = v0 & 0x3f;
```

```
int green = v1 & 0x1f;
int blue = v2 & 0x1f;
int scale = v3 & 0x1f;
// Extract high-order bits, which may be assigned depending on mode
int x0 = (v1 » 6) & 1; int x1 = (v1 » 5) & 1;
int x2 = (v2 » 6) & 1; int x3 = (v2 » 5) & 1;
int x4 = (v3 » 7) & 1; int x5 = (v3 » 6) & 1; int x6 = (v3 » 5) & 1;
// Now move the high-order xs into the right place.
int ohm = 1 « mode;
if( ohm & 0x30 ) green |= x0 « 6;
if( ohm & 0x3A ) green |= x1 « 5;
if( ohm & 0x30 ) blue |= x2 « 6;
if( ohm & 0x3A ) blue |= x3 « 5;
if( ohm & 0x3D ) scale |= x6 « 5;
if( ohm & 0x2D ) scale |= x5 « 6;
if( ohm & 0x04 ) scale |= x4 « 7;
if( ohm & 0x3B ) red |= x4 « 6;
if( ohm & 0x04 ) red |= x3 « 6;
if( ohm & 0x10 ) red |= x5 « 7;
if( ohm & 0x0F ) red |= x2 « 7;
if( ohm & 0x05 ) red |= x1 « 8;
if( ohm & 0x0A ) red |= x0 « 8;
if( ohm & 0x05 ) red |= x0 « 9;
if( ohm & 0x02 ) red |= x6 « 9;
if( ohm & 0x01 ) red |= x3 « 10;
if( ohm & 0x02 ) red |= x5 « 10;
// Shift the bits to the top of the 12-bit result.
static const int shamts[6] = { 1,1,2,3,4,5 };
int shamt = shamts[mode];
red «= shamt; green «= shamt; blue «= shamt; scale «= shamt;
// Minor components are stored as differences
```

if(mode != 5) { green = red - green; blue = red - blue; }

// Swizzle major component into place

if(majcomp == 1) swap(red, green);

if(majcomp == 2) swap(red, blue);

// Clamp output values, set alpha to 1.0

e1.r = clamp(red, 0, 0xFFF);

e1.g = clamp(green, 0, 0xFFF);

e1.b = clamp(blue, 0, 0xFFF);

e1.alpha = 0x780;

e0.r = clamp(red - scale, 0, 0xFFF);

e0.g = clamp(green - scale, 0, 0xFFF);

e0.b = clamp(blue - scale, 0, 0xFFF);

e0.alpha = 0x780;

HDR Endpoint Mode 11 (HDR RGB, Direct)

Mode 11 specifies two RGB values, which it calculates from a number of bitfields (a, b0, b1, c, d0 and d1) which are packed together with some mode bits into the six values (v0, v1, v2, v3, v4, v5):

HDR Mode 11 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	a[7:0]							
V1	m[0]	a[8]	c[5:0]					
V2	m[1]	X0	b0[5:0]					
V3	m[2]	X1	b1[5:0]					
V4	mj[0]	X2	X4	d0[4:0]				
V5	mj[1]	X3	X5	d1[4:0]				

If the major component bits mj[1:0] = b11, then the RGB values are specified directly as

HDR Mode 11 Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V0	R0[11:4]							
V1	R1[11:4]							
V2	G0 [11:4]							
V3	G1[11:4]							
V4	1	B0[11:5]						
V5	1	B1[11:5]						

The mode bits m[2:0] specify the bit allocation for the different values, and the destinations of the extra bits X0 to X5:

Endpoint Bit Mode

Mode	Number of bits				Description of Extra Bits					
	a	b	c	d	X0	X1	X2	X3	X4	X5
0	9	7	6	7	b0[6]	b1[6]	d0[6]	d1[6]	d0[5]	d1[5]
1	9	8	6	6	b0[6]	b1[6]	b0[7]	b1[7]	d0[5]	d1[5]
2	10	6	7	7	a[9]	c[6]	d0[6]	d1[6]	d0[5]	d1[5]
3	10	7	7	6	b0[6]	b1[6]	a[9]	c[6]	d0[5]	d1[5]
4	11	8	6	5	b0[6]	b1[6]	b0[7]	b1[7]	a[9]	a[10]
5	11	6	7	6	a[9]	a[10]	c[7]	c[6]	d0[5]	d1[5]
6	12	7	7	5	b0[6]	b1[6]	a[11]	c[6]	a[9]	a[10]
7	12	6	7	6	a[9]	a[10]	a[11]	c[6]	d0[5]	d1[5]

The complete decoding procedure is as follows:

```
// Find major component
int majcomp = ((v4 & 0x80) » 7) | ((v5 & 0x80) » 6);
// Deal with simple case first
if( majcomp == 3 )
{
e0 = (v0 « 4, v2 « 4, (v4 & 0x7f) « 5, 0x780);
e1 = (v1 « 4, v3 « 4, (v5 & 0x7f) « 5, 0x780);
return;
}
// Decode mode, parameters.
```

```
int mode = ((v1 & 0x80) » 7) | ((v2 & 0x80) » 6) | ((v3 & 0x80) » 5);
int va = v0 | ((v1 & 0x40) « 2);
int vb0 = v2 & 0x3f;
int vb1 = v3 & 0x3f;
int vc = v1 & 0x3f;
int vd0 = v4 & 0x7f;
int vd1 = v5 & 0x7f;
// Assign top bits of vd0, vd1.
static const int dbitstab[8] = {7,6,7,6,5,6,5,6};
vd0 = signextend( vd0, dbitstab[mode] );
vd1 = signextend( vd1, dbitstab[mode] );
// Extract and place extra bits
int x0 = (v2 » 6) & 1;
int x1 = (v3 » 6) & 1;
int x2 = (v4 » 6) & 1;
int x3 = (v5 » 6) & 1;
int x4 = (v4 » 5) & 1;
int x5 = (v5 » 5) & 1;
int ohm = 1 « mode;
if( ohm & 0xA4 ) va |= x0 « 9;
if( ohm & 0x08 ) va |= x2 « 9;
if( ohm & 0x50 ) va |= x4 « 9;
if( ohm & 0x50 ) va |= x5 « 10;
if( ohm & 0xA0 ) va |= x1 « 10;
if( ohm & 0xC0 ) va |= x2 « 11;
if( ohm & 0x04 ) vc |= x1 « 6;
if( ohm & 0xE8 ) vc |= x3 « 6;
if( ohm & 0x20 ) vc |= x2 « 7;
if( ohm & 0x5B ) vb0 |= x0 « 6;
if( ohm & 0x5B ) vb1 |= x1 « 6;
if( ohm & 0x12 ) vb0 |= x2 « 7;
```

```
if( ohm & 0x12 ) vb1 |= x3 « 7;
// Now shift up so that major component is at top of 12-bit value
int shamt = (modeval » 1) ^ 3;
va «= shamt; vb0 «= shamt; vb1 «= shamt;
vc «= shamt; vd0 «= shamt; vd1 «= shamt;
e1.r = clamp( va, 0, 0xFFF );
e1.g = clamp( va - vb0, 0, 0xFFF );
e1.b = clamp( va - vb1, 0, 0xFFF );
e1.alpha = 0x780;
e0.r = clamp( va - vc, 0, 0xFFF );
e0.g = clamp( va - vb0 - vc - vd0, 0, 0xFFF );
e0.b = clamp( va - vb1 - vc - vd1, 0, 0xFFF );
e0.alpha = 0x780;
if( majcomp == 1 )
{
swap( e0.r, e0.g ); swap( e1.r, e1.g );
}
else if( majcomp == 2 )
{
swap( e0.r, e0.b ); swap( e1.r, e1.b );
}
```

HDR Endpoint Mode 14 (HDR RGB, Direct + LDR Alpha)

Mode 14 specifies two RGBA values, using the eight values (v0, v1, v2, v3, v4, v5, v6, v7). First, the RGB values are decoded from (v0..v5) using the method from Mode 11. Then the alpha values are filled in from v6 and v7:

```
// Decode RGB as for mode 11
(e0,e1) = decode_mode_11(v0,v1,v2,v3,v4,v5)
// Now fill in the alphas
e0.alpha = v6;
e1.alpha = v7;
```

HDR Endpoint Mode 15 (HDR RGB, Direct + HDR Alpha)

Mode 15 specifies two RGBA values, using the eight values (v0, v1, v2, v3, v4, v5, v6, v7). First, the RGB values are decoded from (v0..v5) using the method from Mode 11. The alpha values are stored in values v6 and v7 as a mode and two values which are interpreted according to the mode:

HDR Mode 15 Alpha Value Layout

Value	Bit[7]	Bit[6]	Bit[5]	Bit[4]	Bit[3]	Bit[2]	Bit[1]	Bit[0]
V6	M0	A[6:0]						
V7	M1	B[6:0]						

The alpha values are decoded from v6 and v7 as follows:

```
// Decode RGB as for mode 11
(e0,e1) = decode_mode_11(v0,v1,v2,v3,v4,v5)
// Extract mode bits
mode = ((v6 » 7) & 1) | ((v7 » 6) & 2);
v6 &= 0x7F;
v7 &= 0x7F;
if(mode==3)
{
// Directly specify alphas
e0.alpha = v6 « 5;
e1.alpha = v7 « 5;
}
else
{
// Transfer bits from v7 to v6 and sign extend v7.
v6 |= (v7 « (mode+1))) & 0x780;
v7 &= (0x3F » mode);
v7 ^= 0x20 » mode;
v7 -= 0x20 » mode;
v6 «= (4-mode);
v7 «= (4-mode);
// Add delta and clamp
```

```
v7 += v6;
v7 = clamp(v7, 0, 0xFFF);
e0.alpha = v6;
e1.alpha = v7;
}
```

Restrictions on Number of Partitions Per Block

Following table gives total number of partitions for each CEM mode given the restriction of total up to 16 integer values being decoded from the Integer Sequence Coding sequence.

Groups	Max Number of Partition	CEM Modes
(v0,v1)	4	0,1,2,3
(v0,v1,v2,v3)	4	4,5,6,7
(v0,v1,v2,v3,v4,v5)	3	8,9,10,11
(v0,v1,v2,v3,v4,v5,v6,v7)	2	12,13,14,15

Index Decoding

The index information is stored as a stream of bits, growing downwards from the most significant bit in the block. Bit *n* in the stream is thus bit 127-*n* in the block.

For each location in the index grid, a value (in the specified range) is packed into the stream. These are ordered in a raster pattern starting from location (0,0,0), with the X dimension increasing fastest, and the Z dimension increasing slowest. If dual-plane mode is selected, both indices are emitted together for each location, plane 0 first, then plane 1.

Index Unquantization

Each index plane is specified as a sequence of integers in a given range. These values are packed using integer sequence encoding.

Once unpacked, the values must be unquantized from their storage range, returning them to a standard range of 0..64. The procedure for doing so is similar to the color endpoint unquantization.

First, we unquantize the actual stored index values to the range 0..63.

For bit-only representations, this is simple bit replication from the most significant bit of the value.

For trit or quint-based representations, this involves a set of bit manipulations and adjustments to avoid the expense of full-width multipliers.

For representations with no additional bits, the results are as follows:

Index Unquantization Values

Range	0	1	2	3	4
0..2	0	32	63	-	-
0..4	0	16	32	47	63

For other values, we calculate the initial inputs to a bit manipulation procedure. These are denoted A, B, C and D and are decoded using the range as follows:

Index Unquantization Parameters

Range	Trits	Quints	Bits	Bit value	A (7 bits)	B (7 bits)	C (7 bits)	D (3 bits)
0..5	1		1	a	aaaaaaa	0000000	50	Trit
0..9		1	1	a	aaaaaaa	0000000	28	Quint
0..11	1		2	ba	aaaaaaa	b000b0b	23	Trit
0..19		1	2	ba	aaaaaaa	b0000b0	13	Quint
0..23	1		3	cba	aaaaaaa	cb000cb	11	Trit

These are then processed as follows:

T = D * C + B;

T = T ^ A;

T = (A & 0x20) | (T » 2);

The multiply in the first line is nearly trivial as it only needs to multiply by 0, 1, 2, 3 or 4. As a final step, for all types of value, the range is expanded from 0..63 up to 0..64 as follows:

if (T > 32) { T += 1; }

This allows the implementation to use 64 as a divisor during interpolation, which is much easier than using 63.

Infill Process

After unquantization, the indexes are subject to index selection and infill. The infill method is used to calculate the index for a texel position, based on the indices in the stored index grid array (which may be a different size). The procedure below must be followed exactly, to ensure bit exact results. The block size is specified as three dimensions along the s, t and r axes (Bs, Bt, Br). Texel coordinates within the block (s,t,r) can have values from 0 to one less than the block dimension in that axis.

For each block dimension, we compute scale factors (Ds, Dt, Dr)

Ds = floor((1024 + floor(Bs/2)) / (Bs-1));

Dt = floor((1024 + floor(Bt/2)) / (Bt-1));

Dr = floor((1024 + floor(Br/2)) / (Br-1));

Since the block dimensions are constrained, these are easily looked up in a table. These scale factors are then used to scale the (s,t,r) coordinates to a homogeneous coordinate (cs, ct, cr):

cs = Ds * s;

ct = Dt * t;

cr = Dr * r;

This homogeneous coordinate (cs, ct, cr) is then scaled again to give a coordinate (gs, gt, gr) in the index-grid space . The index-grid is of size (N, M, Q), as specified in the index mode field:

gs = (cs*(N-1)+32) » 6;

gt = (ct*(M-1)+32) » 6;

gr = (cr*(Q-1)+32) » 6;

The resulting coordinates may be in the range 0..176. These are interpreted as 4:4 unsigned fixed point numbers in the range 0.0 .. 11.0. If we label the integral parts of these (js, jt, jr) and the fractional parts (fs, ft, fr), then:

js = gs » 4; fs = gs & 0x0F;

jt = gt » 4; ft = gt & 0x0F;

jr = gr » 4; fr = gr & 0x0F;

These values are then used to interpolate between the stored indices. This process differs for 2D and 3D.

For 2D, bilinear interpolation is used:

v0 = js + jt*N;

p00 = decode_index(v0);

p01 = decode_index(v0 + 1);

p10 = decode_index(v0 + N);

p11 = decode_index(v0 + N + 1);

The function decode_index(n) decodes the nth index in the stored index stream. The values p00 to p11 are the indices at the corner of the square in which the texel position resides. These are then weighted using the fractional position to produce the effective index i as follows:

w11 = (fs*ft+8) » 4;

w10 = ft – w11;

w01 = fs – w11;

w00 = 16 – fs – ft + w11;

i = (p00*w00 + p01*w01 + p10*w10 + p11*w11 + 8) » 4;

For 3D, simplex interpolation is used as it is cheaper than a naïve trilinear interpolation. First, we pick some parameters for the interpolation based on comparisons of the fractional parts of the texel position:

fs>ft	ft>fr	fs>fr	s1	s2	w0	w1	w2	w3
True	True	True	1	N	16-fs	fs-ft	ft-fr	fr
False	True	True	N	1	16-ft	ft-fs	fs-fr	fr
True	False	True	1	N*M	16-fs	fs-fr	fr-ft	ft
True	False	False	N*M	1	16-fr	fr-fs	fs-ft	ft
False	True	False	N	N*M	16-ft	ft-fr	fr-fs	fs
False	False	False	N*M	N	16-fr	fr-ft	ft-fs	fs

The effective index *i* is then calculated as:

v0 = js + jt*N + jr*N*M;

p0 = decode_index(v0);

p1 = decode_index(v0 + s1);

p2 = decode_index(v0 + s1 + s2);

p3 = decode_index(v0 + N*M + N + 1);

i = (p0*w0 + p1*w1 + p2*w2 + p3*w3 + 8) » 4;

Index Application

Once the effective index *i* for the texel has been calculated, the color endpoints are interpolated and expanded. For LDR endpoint modes, each color component C is calculated from the corresponding 8-bit endpoint components C0 and C1 as follows:

If sRGB conversion is not enabled, C0 and C1 are first expanded to 16 bits by bit replication:

C0 = (C0 « 8) | C0; C1 = (C1 « 8) | C1;

If sRGB conversion is enabled, C0 and C1 are expanded to 16 bits differently, as follows:

C0 = (C0 « 8) | 0x80; C1 = (C1 « 8) | 0x80;

C0 and C1 are then interpolated to produce a UNORM16 result C:

C = floor((C0*(64-i) + C1*i + 32)/64)

If sRGB conversion is enabled, the top 8 bits of the interpolation result are passed to the external sRGB conversion block. Otherwise, if C = 65535, then the final result is 1.0 (0x3C00) otherwise C is divided by 216 and the infinite-precision result of the division is converted to FP16 with round-to-zero semantics. For HDR endpoint modes, color values are represented in a 12-bit logarithmic representation, and interpolation occurs in a piecewise-approximate logarithmic manner as follows:

In LDR mode, the error result is returned.

In HDR mode, the color components from each endpoint, C0 and C1, are initially shifted left 4 bits to become 16-bit integer values and these are interpolated in the same way as LDR. The 16-bit value C is then decomposed into the top five bits, E, and the bottom 11 bits M, which are then processed and recombined with E to form the final value Cf:

C = floor((C0*(64-i) + C1*i + 32)/64)

E = (C&0xF800) » 11; M = C&0x7FF;

if (M < 512) { Mt = 3*M; }

else if (M >= 1536) { Mt = 5*M – 2048; }

else { Mt = 4*M – 512; }

Cf = (E«10) + (Mt»3)

This final value Cf is interpreted as an IEEE FP16 value. If the result is +Inf or NaN, it is converted to the bit pattern 0x7BFF, which is the largest representable finite value.

Dual-Plane Decoding

If dual-plane mode is disabled, all of the endpoint components are interpolated using the same index value. If dual-plane mode is enabled, two indices are stored with each texel. One component is then selected to use the second index for interpolation, instead of the first index. The first index is then used for all other components.

The component to treat specially is indicated using the 2-bit Color Component Selector (CCS) field as follows:

Dual Plane Color Component Selector Values

Value	Index 0	Index 1
0	GBA	R
1	RBA	G
2	RGA	B
3	RGB	A

The CCS bits are stored at a variable position directly below the index bits and any additional CEM bits.

Partition Pattern Generation

When multiple partitions are active, each texel position is assigned a partition index. This partition index is calculated using a seed (the partition pattern index), the texel's x,y,z position within the block, and the number of partitions. An additional argument, small_block, is set to 1 if the number of texels in the block is less than 31, otherwise it is set to 0. The full partition selection algorithm is as follows:

```
int select_partition(int seed, int x, int y, int z
int partitioncount, int small_block)
{
if( small_block ){ x «= 1; y «= 1; z «= 1; }
seed += (partitioncount-1) * 1024;
uint32_t rnum = hash52(seed);
uint8_t seed1 = rnum & 0xF;
uint8_t seed2 = (rnum » 4) & 0xF;
uint8_t seed3 = (rnum » 8) & 0xF;
uint8_t seed4 = (rnum » 12) & 0xF;
uint8_t seed5 = (rnum » 16) & 0xF;
uint8_t seed6 = (rnum » 20) & 0xF;
uint8_t seed7 = (rnum » 24) & 0xF;
uint8_t seed8 = (rnum » 28) & 0xF;
uint8_t seed9 = (rnum » 18) & 0xF;
uint8_t seed10 = (rnum » 22) & 0xF;
uint8_t seed11 = (rnum » 26) & 0xF;
uint8_t seed12 = ((rnum » 30) | (rnum « 2)) & 0xF;
```

```
seed1 *= seed1; seed2 *= seed2; seed3 *= seed3; seed4 *= seed4;
seed5 *= seed5; seed6 *= seed6; seed7 *= seed7; seed8 *= seed8;
seed9 *= seed9; seed10 *= seed10; seed11 *= seed11; seed12 *= seed12;
int sh1, sh2, sh3;
if( seed & 1 )
{ sh1 = (seed & 2 ? 4 : 5); sh2 = (partitioncount == 3 ? 6 : 5); }
else
{ sh1 = (partitioncount == 3 ? 6 : 5); sh2 = (seed & 2 ? 4 : 5); }
sh3 = (seed & 0x10) ? sh1 : sh2:
seed1 »= sh1; seed2 »= sh2; seed3 »= sh1; seed4 »= sh2;
seed5 »= sh1; seed6 »= sh2; seed7 »= sh1; seed8 »= sh2;
seed9 »= sh3; seed10 »= sh3; seed11 »= sh3; seed12 »= sh3;
int a = seed1*x + seed2*y + seed11*z + (rnum » 14);
int b = seed3*x + seed4*y + seed12*z + (rnum » 10);
int c = seed5*x + seed6*y + seed9 *z + (rnum » 6);
int d = seed7*x + seed8*y + seed10*z + (rnum » 2);
a &= 0x3F; b &= 0x3F; c &= 0x3F; d &= 0x3F;
if( partitioncount < 4 ) d = 0;
if( partitioncount < 3 ) c = 0;
if( a >= b && a >= c && a >= d ) return 0;
else if( b >= c && b >= d ) return 1;
else if( c >= d ) return 2;
else return 3;
}
```

As has been observed before, the bit selections are much easier to express in hardware than in C.

The seed is expanded using a hash function hash52, which is defined as follows:

```
uint32_t hash52( uint32_t p )
{
p ^= p » 15; p -= p « 17; p += p « 7; p += p « 4; p ^= p » 5;
p += p « 16; p ^= p » 7; p ^= p » 3; p ^= p « 6; p ^= p » 17;
return p;
```

}

This assumes that all operations act on 32-bit values

Data Size Determination

The size of the data used to represent color endpoints is not explicitly specified. Instead, it is determined from the index mode and number of partitions as follows:

config_bits = 17;

if(num_partitions>1)

if(single_CEM)

config_bits = 29;

else

config_bits = 24 + 3*num_partitions;

num_indices = M * N * Q; // size of index grid

if(dual_plane)

config_bits += 2;

num_indices *= 2;

index_bits = ceil(num_indices*8*trits_in_index_range/5) +

ceil(num_indices*7*quints_in_index_range/3) +

num_indices*bits_in_index_range;

remaining_bits = 128 – config_bits – index_bits;

num_CEM_pairs = base_CEM_class+1 + count_bits(extra_CEM_bits);

The CEM value range is then looked up from a table indexed by remaining bits and num_CEM_pairs. This table is initialized such that the range is as large as possible, consistent with the constraint that the number of bits required to encode num_CEM_pairs pairs of values is not more than the number of remaining bits.

An equivalent iterative algorithm would be:

num_CEM_values = num_CEM_pairs*2;

for(range = each possible CEM range in descending order of size)

{

CEM_bits = ceil(num_CEM_values*8*trits_in_CEM_range/5) +

ceil(num_CEM_values*7*quints_in_CEM_range/3) +

num_CEM_values*bits_in_CEM_range;

if(CEM_bits <= remaining_bits)

break;

}

return range;

In cases where this procedure results in unallocated bits, these bits are not read by the decoding process and can have any value.

3D Void-Extent Blocks

The layout of a 3D Void-Extent block is as follows:

127:112	111:96	95:80	79:64	63:55	54:46	45:37	36:28	27:19	18:10	9:9	8:0
A	B	G	R	P_high	P_low	T_high	T_low	S_high	S_low	D	111111100

Bit 9 is the Dynamic Range flag, which indicates the format in which colors are stored. Value 0 indicates LDR, in which case the color components are stored as UNORM16 values, while value 1 indicates HDR, in which case the color components are stored as FP16 values.

The reason for the storage of UNORM16 values in the LDR case is due to the possibility that the value will need to be passed on to sRGB conversion. By storing the color value in the format which comes out of the interpolator, before the conversion to FP16, we avoid having to have separate versions for sRGB and linear modes.

If a void-extent block with HDR values is decoded in LDR mode, then the result will be the error color, opaque magenta, for all texels within the block.

The minimum and maximum coordinate values are treated as unsigned integers and then normalized into the range 0..1 (by dividing by 2^{13}-1 or 2^{9}-1, for 2D and 3D respectively). The maximum values for each dimension must be greater than the corresponding minimum values, unless they are all all-1s. If all the coordinates are all-1s, then the void extent is ignored, and the block is simply a constant-color block.

Illegal Encodings

In ASTC, there is a variety of ways to encode an illegal block. Decoders are required to recognize all illegal blocks and emit the standard Error Block color value upon encountering an illegal block. The standard Error Block color value is opaque magenta (R, G, B, A) = (0xFF, 0x00, 0xFF, 0xFF) in the LDR operation mode, and a vector of NaNs (**R, G, B, A**)=(NaN, NaN, NaN, NaN) in the HDR operation mode. It is recommended that the NaN be encoded as the bit-pattern 0xFFFF.

Here is a comprehensive list of situations that represent illegal block encodings:

- The index bit mode specified is one of the modes explicitly listed as Reserved.
- An index bit mode has been specified that would require more than 64 indexes total.
- An index bit mode has been specified that would require more than 96 bits for the Index Integer Sequence Encoding.

- An index bit mode has been specified that would require fewer than 24 bits for the Index Integer Sequence Encoding.
- The size of the index grid exceeds the size of the block footprint in any dimension.
- Color endpoint modes have been specified such that the Color Integer Sequence Encoding would require more than 18 integers.
- The number of bits available for color endpoint encoding after all the other fields have been counted is less than *ceil(13C/5)* where C is the number of color endpoint integers (this would restrict color integers to a range smaller than 0..5, which is not supported).
- Dual Index Mode is enabled for a block with 4 partitions.
- Void-Extent blocks where the low coordinate for some texture axis is greater than or equal to the high coordinate.
- Under 3D mode, the depth (Q) is not 1

In LDR mode, a block which has both HDR and LDR endpoint modes assigned to different partitions is not an error block. Only those texels which belong to the HDR partition will result in the error color. Texels belonging to a LDR partition will be decoded as normal.

Profile Support

In order to ease verification and accelerate adoption, an LDR-only subset of the full ASTC specification has been made available.

Implementations of this LDR Profile must satisfy the following requirements:

- All textures with valid encodings for LDR Profile must decode identically using either a LDR Profile or Full Profile decoder.
- All features included only in the Full Profile must be treated as reserved in the LDR Profile, and return the error color on decoding.
- Any sequence of API calls valid for the LDR Profile must also be valid for the Full Profile and return identical results when given a texture encoded for the LDR Profile.

The feature subset for the LDR profile is:

- 2D textures only.
- Only those block sizes listed in Table 5 are supported.
- LDR operation mode only.
- Only LDR endpoint formats must be supported namely formats 0, 1, 4, 5, 6, 8, 9, 10, 12, 13.
- Decoding from a HDR endpoint results in the error color.
- Interpolation returns UNORM8 results when used in conjunction with sRGB.
- LDR void extent blocks must be supported, but void extents may not be checked.

Video Pixel/Texel Formats

This section describes the “video” pixel/texel formats with respect to memory layout. See the Overlay chapter for a description of how the Y, U, V components are sampled.

Packed Memory Organization

Color components are all 8 bits in size for YUV formats. For YUV 4:2:2 formats each DWord will contain two pixels and only the byte order affects the memory organization.

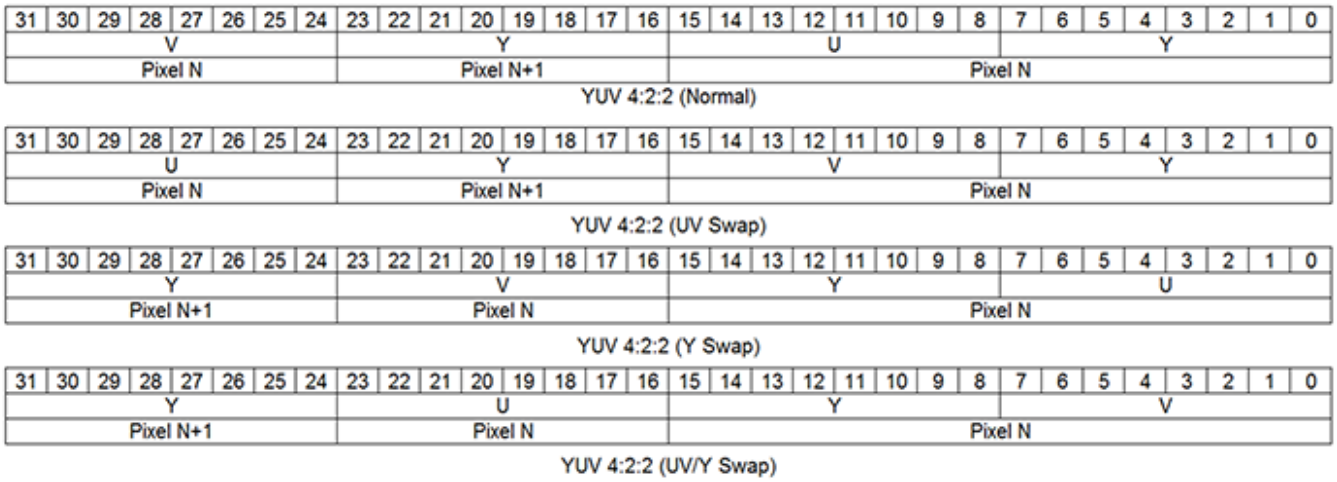
The following four YUV 4:2:2 surface formats are supported, listed with alternate names:

- YCRCB_NORMAL (YUYV/YUY2)
- YCRCB_SWAPUVY (VYUY) (R8G8_B8G8_UNORM)
- YCRCB_SWAPUV(YVYU) (G8R8_G8B8_UNORM)
- YCRCB_SWAPY (UYVY)

The channels are mapped as follows:

Cr (V)	Red
Y	Green
Cb (U)	Blue

Memory layout of packed YUV 4:2:2 formats



Planar Memory Organization

Planar formats use what could be thought of as separate buffers for the three color components. Because there is a separate stride for the Y and U/V data buffers, several memory footprints can be supported.

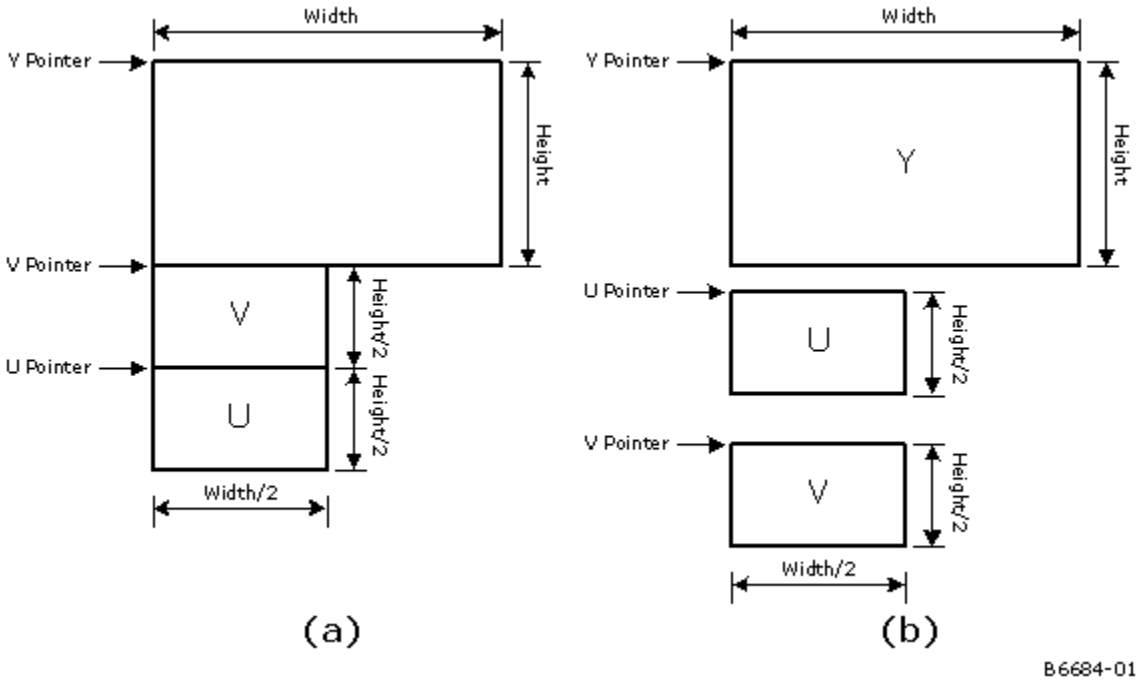
Note: There is no direct support for use of planar video surfaces as textures. The sampling engine can be used to operate on each of the 8bpp buffers separately (via a single-channel 8-bit format such as I8_UNORM). The U and V buffers can be written concurrently by using multiple render targets from the pixel shader. The Y buffer must be

written in a separate pass due to its different size.

The following figure shows two types of memory organization for the YUV 4:2:0 planar video data:

1. The memory organization of the common YV12 data, where all three planes are contiguous and the strides of U and V components are half of that of the Y component.
2. An alternative memory structure that the addresses of the three planes are independent but satisfy certain alignment restrictions.

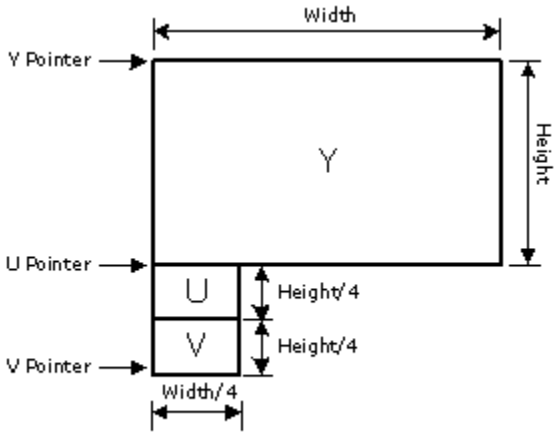
YUV 4:2:0 Format Memory Organization



The following figure shows memory organization of the planar YUV 4:1:0 format where the planes are contiguous.

Note: The chroma planes (U and V), when separate (case b above) are treated as half-pitch with respect to the Y plane. In general, YV12 is supported only in linear format because separate planes cannot be supported correctly with a tiled format.

YUV 4:1:0 Format Memory Organization



B6685-01

The table below shows how position within a Planar YUV surface chroma plane is calculated for various cases ot U and V pitch and position. It also shows restrictions on the alignment of the planes in memory when Y Height is a multiple of 4 or when Interleaved Chroma (e.g. NV21) is used.

Case	Interleave Chroma	Pitch	Vertical U/V Offset	Restrictions
YUV with Half Pitch Chroma	No	Half	When U is below Y Y_Uoffset = Y_Height * 2 Y_Voffset = Y_Height * 2 + V_Height When V is below Y Y_Uoffset = Y_Height * 2 + V_Height Y_Voffset = Y_Height * 2	(Y Height)%4 = 0 (U Height)%2 = 0 (V Height)%2 = 0 Vertical for Y surface must be 0
YUV with Full Pitch Chroma	Yes	Full	When U is below Y Y_Uoffset = Y_Height Y_Voffset = Y_Height + V_Height When V is below Y Y_Uoffset = Y_Height + V_Height Y_Voffset = Y_Height	(Y Height)%2 = 0 (U Height)%2 = 0 (V Height)%2 = 0
YUV for Media Sampling	Yes	Always Full	Same as 3D full pitch	Same as 3D full pitch

Raw Format

A format called "RAW" is available that is only supported with the untyped surface read/write, block, scattered, and atomic operation data port messages. It means that the surface has no inherent format. Surfaces of type RAW are addressed with byte-based offsets. The RAW surface format can be applied only to surface types of BUFFER and STRBUF.

Surface Memory Organizations

See *Memory Interface Functions* chapter for a discussion of tiled vs. linear surface formats.

Display, Overlay, Cursor Surfaces

These surfaces are memory image buffers (planes) used to refresh a display device in non-VGA mode. See the Display chapter for specifics on how these surfaces are defined/used.

2D Render Surfaces

These surfaces are used as general source and/or destination operands in 2D BLT operations.

Note that there is no coherency between 2D render surfaces and the texture cache. Software must explicitly invalidate the texture cache before using a texture that has been modified via the BLT engine.

See the 2D Instruction and 2D Rendering chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

2D Monochrome Source

These 1 BPP (bit per pixel) surfaces are used as source operands to certain 2D BLT operations, where the BLT engine expands the 1 BPP source to the required color depth.

The texture cache stores any monochrome sources. There is no mechanism to maintain coherency between 2D render surfaces and texture-cached monochrome sources. Software must explicitly invalidate the texture cache before using a memory-based monochrome source that has been modified via the BLT engine. (Here the assumption is that SW enforces memory-based monochrome source surfaces as read-only surfaces.)

See the 2D Instruction and 2D Rendering chapters for specifics on how these surfaces are used, restrictions on their size, placement, coherency rules, etc.

2D Color Pattern

Color pattern surfaces are used as special pattern operands in 2D BLT operations.

The device uses the texture cache to store color patterns. There is no mechanism to maintain coherency between 2D render surfaces and (texture)-cached color patterns. Software is required to explicitly invalidate the texture cache before using a memory-based color pattern that has been modified via the BLT engine. (Here the assumption is that SW enforces memory-based color pattern surfaces as read-only surfaces.)

See the *2D Instruction* and *2D Rendering* chapters for specifics on how these surfaces are used, restrictions on their size, placement, etc.

3D Color Buffer (Destination) Surfaces

3D Color Buffer surfaces hold per-pixel color values for use in the 3D Pipeline. The 3D Pipeline always requires a Color Buffer to be defined.

See the Non-Video Pixel/Texel Formats section in this chapter for details on the Color Buffer pixel formats. See the 3D Instruction and 3D Rendering chapters for Color Buffer usage details.

The Color Buffer is defined as the BUFFERID_COLOR_BACK memory buffer via the 3DSTATE_BUFFER_INFO instruction. That buffer can be mapped to LM or SM (snooped or unsnooped), and can be linear or tiled. When both the Depth and Color Buffers are tiled, the respective Tile Walk directions must match.

When a linear Color Buffer and a linear Depth Buffer are used together:

- The buffers may have different pitches, though both pitches must be a multiple of 32 bytes.
- The buffers must be co-aligned with a 32-byte region.

3D Depth Buffer Surfaces

Depth Buffer surfaces hold per-pixel depth values and per-pixel stencil values for use in the 3D Pipeline. The 3D Pipeline does not require a Depth Buffer in general, though a Depth Buffer is required to perform non-trivial Depth Test and Stencil Test operations.

The Depth Buffer is specified via the 3DSTATE_DEPTH_BUFFER command. See the description of that instruction in *Windower* for restrictions.

See *Depth Buffer Formats* below for a summary of the possible depth buffer formats. See the Depth Buffer Formats section in this chapter for details on the pixel formats. See the *Windower* and *DataPort* chapters for details on the usage of the Depth Buffer.

Depth Buffer Formats

DepthBufferFormat / DepthComponent	BPP (Bits Per Pixel)	Description
D32_FLOAT_S8X24_UINT	64	32-bit floating point Z depth value in first DWord, 8-bit stencil in lower byte of second DWord
D32_FLOAT	32	32-bit floating point Z depth value
D24_UNORM_S8_UINT	32	24-bit fixed point Z depth value in lower 3 bytes, 8-bit stencil value in upper byte
D16_UNORM	16	16-bit fixed point Z depth value

3D Separate Stencil Buffer Surfaces

Separate Stencil Buffer surfaces hold per-pixel stencil values for use in the 3D Pipeline. Note that the 3D Pipeline does not require a Stencil Buffer to be allocated, though a Stencil Buffer is required to perform non-trivial Stencil Test operations.

UNRESOLVED CROSS REFERENCE, Depth Buffer Formats summarizes Stencil Buffer formats. Refer to the Stencil Buffer Formats section in this chapter for details on the pixel formats. Refer to the *Windower* chapters for Stencil Buffer usage details.

The Stencil buffer is specified via the 3DSTATE_STENCIL_BUFFER command. See that instruction description in *Windower* for restrictions.

Depth Buffer Formats

DepthBufferFormat / DepthComponent	BPP (bits per pixel)	Description
R8_ UNIT	8	8-bit stencil value in a byte

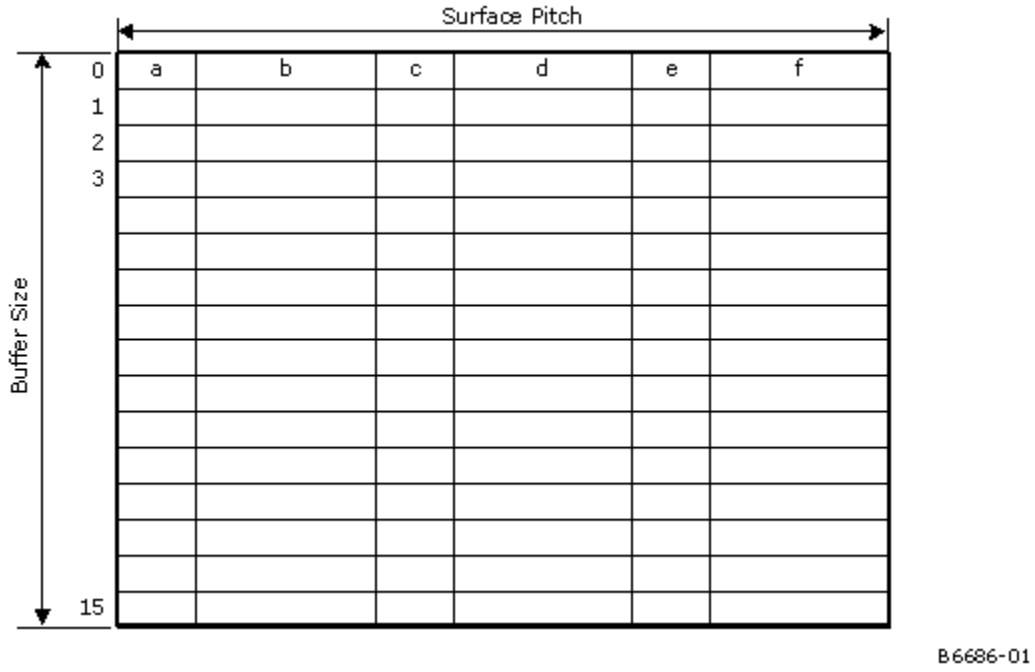
Surface Layout

In addition to restrictions on maximum height, width, and depth, surfaces are also restricted to a maximum size in bytes. This maximum is 2 GB for all products and all surface types.

Buffers

A buffer is an array of structures. Each structure contains up to 2048 bytes of elements. Each element is a single surface format using one of the supported surface formats depending on how the surface is being accessed. The surface pitch state for the surface specifies the size of each structure in bytes.

The buffer is stored in memory contiguously with each element in the structure packed together, and the first element in the next structure immediately following the last element of the previous structure. Buffers are supported only in linear memory.



Structured Buffers

A structured buffer is a surface type that is accessed by a 2-dimensional coordinate. It can be thought of as an array of structures, where each structure is a predefined number of DWords in size. The first coordinate (U) defines the array index, and the second coordinate (V) is a byte offset into the structure which must be a multiple of 4 (DWord-aligned). A structured buffer must be defined with **Surface Format** RAW.

The structured buffer has only one dimension programmed in SURFACE_STATE which indicates the array size. The byte offset dimension (V) is assumed to be bounded only by the **Surface Pitch**.

1D Surfaces

One-dimensional surfaces are identical to 2D surfaces with height of one. Arrays of 1D surfaces are also supported. Please refer to the 2D Surfaces section for details on how these surfaces are stored.

2D Surfaces

Surfaces that comprise texture mip-maps are stored in a fixed "monolithic" format and referenced by a single base address. The base map and associated mipmaps are located within a single rectangular area of memory identified by the base address of the upper left corner and a pitch. The base address references the upper left corner of the base map. The pitch must be specified at least as large as the widest mip-map. In some cases it must be wider; see the section on Minimum Pitch below.

These surfaces may be overlapped in memory and must adhere to the following memory organization rules:

- For non-compressed texture formats, each mipmap must start on an even row within the monolithic rectangular area. For 1-texel-high mipmaps, this may require a row of padding below the previous mipmap. This restriction does not apply to any compressed texture formats; each subsequent (lower-res) compressed mipmap is positioned directly below the previous mipmap.
- Vertical alignment restrictions vary with memory tiling type: 1 DWord for linear, 16-byte (DQWord) for tiled. (Note that tiled mipmaps are *not* required to start at the left edge of a tile row.)

Computing MIP Level Sizes

Map width and height specify the size of the largest MIP level (LOD 0). Less detailed LOD level (i+1) sizes are determined by dividing the width and height of the current (i) LOD level by 2 and truncating to an integer (floor). This is equivalent to shifting the width/height by 1 bit to the right and discarding the bit shifted off. The map height and width are clamped on the low side at 1.

In equations, the width and height of an LOD "*L*" can be expressed as:

W_L = *((width » L) > 0? width » L:1)*

H_L = *((height » L) > 0? height » L:1)*

If the surface is multisampled and it is a depth or stencil surface or **Multisampled Surface StorageFormat** in SURFACE_STATE is MSFMT_DEPTH_STENCIL, W_L and H_L must be adjusted as follows before proceeding:

Number of Multisamples	W_L =	H_L =
2	ceiling(W_L / 2) * 4	H_L [no adjustment]
4	ceiling(W_L / 2) * 4	ceiling(H_L / 2) * 4
8	ceiling(W_L / 2) * 8	ceiling(H_L / 2) * 4d
16	ceiling(W_L / 2) * 8	ceiling(H_L / 2) * 8

Base Address for LOD Calculation

It is conceptually easier to think of the space that the map uses in Cartesian space (x, y), where x and y are in units of texels, with the upper left corner of the base map at (0, 0). The final step is to convert from Cartesian coordinates to linear addresses as documented at the bottom of this section.

It is useful to think of the concept of "stepping" when considering where the next MIP level will be stored. We either step down or step right when moving to the next higher LOD.

- for MIPLAYOUT_RIGHT maps:
    - step right when moving from LOD 0 to LOD 1
    - step down for all of the other MIPs
- for MIPLAYOUT_BELOW maps:
    - step down when moving from LOD 0 to LOD 1
    - step right when moving from LOD 1 to LOD 2
    - step down for all of the other MIPs

To account for the cache line alignment required, we define *i* and *j* as the width and height, respectively, of an *alignment unit*. This alignment unit is defined below. We then define lower-case w_L and h_L as the padded width and height of LOD "*L*" as follows:

$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

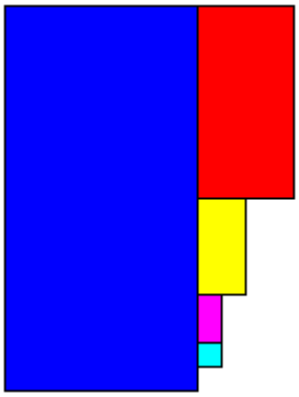
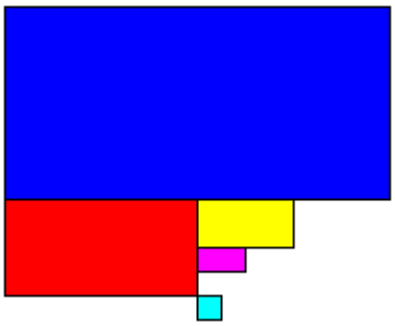
$$h_L = j * ceil\left(\frac{H_L}{j}\right)$$

For separate stencil buffer, the width must be mutiplied by 2 and height divided by 2 as follows:

$$w_l = 2 * i * ceil\left(\frac{W_L}{i}\right)$$

$$h_L = 1/2 * j * ceil\left(\frac{H_L}{j}\right)$$

Equations to compute the upper left corner of each MIP level are then as follows:

for *MIPLAYOUT_RIGHT* maps: $LOD_0 = (0,0)$ $LOD_1 = (w_0,0)$ $LOD_2 = (w_0,h_1)$ $LOD_3 = (w_0,h_1 + h_2)$ $LOD_4 = (w_0,h_1 + h_2 + h_3)$ ...	
for *MIPLAYOUT_BELOW* maps: $LOD_0 = (0,0)$ $LOD_1 = (0,h_0)$ $LOD_2 = (w_1,h_0)$ $LOD_3 = (w_1,h_0 + h_2)$ $LOD_4 = (w_1,h_0 + h_2 + h_3)$ ...	

Minimum Pitch for MIPLAYOUT_RIGHT and Other Maps

For MIPLAYOUT_RIGHT maps, the minimum pitch must be calculated before choosing a fence to place the map within. This is approximately equal to 1.5x the pitch required by the base map, with possible adjustments made for cache line alignment. For MIPLAYOUT_BELOW and MIPLAYOUT_LEGACY maps, the minimum pitch required is equal to that required by the base (LOD 0) map.

A safe but simple calculation of minimum pitch is equal to 2x the pitch required by the base map for MIPLAYOUT_RIGHT maps. This ensures that enough pitch is available, and since it is restricted to MIPLAYOUT_RIGHT maps, not much memory is wasted. It is up to the driver (hardware independent) whether to use this simple determination of pitch or a more complex one.

Cartesian to Linear Address Conversion

A set of variables are defined in addition to the i and j defined above.

- b = bytes per texel of the native map format (0.5 for DXT1, FXT1, and 4-bit surface format, 2.0 for YUV 4:2:2, others aligned to surface format)
- t = texel rows / memory row (4 for DXT1-5 and FXT1, 1 for all other formats)
- p = pitch in bytes (equal to pitch in dwords * 4)
- B = base address in bytes (address of texel 0,0 of the base map)
- x, y = cartestian coordinates from the above calculations in units of texels (assumed that x is always a multiple of i and y is a multiple of j)
- A = linear address in bytes

$$A = B + \frac{yp}{t} + xbt$$

This calculation gives the linear address in bytes for a given MIP level (taking into account L1 cache line alignment requirements).

Compressed Mipmap Layout

Mipmaps of textures using compressed (DXTn, FXT) texel formats are also stored in a monolithic format. The compressed mipmaps are stored in a similar fashion to uncompressed mipmaps, with each block of source (uncompressed) texels represented by a 1 or 2 QWord compressed block. The compressed blocks occupy the same logical positions as the texels they represent, where each row of compressed blocks represent a 4-high row of uncompressed texels. The format of the blocks is preserved, i.e., there is no “intermediate” format as required on some other devices.

The following exceptions apply to the layout of compressed (vs. uncompressed) mipmaps:

- Mipmaps are not required to start on even rows, therefore each successive mip level is located on the texel row immediately below the last row of the previous mip level. Pad rows are neither required nor allowed.
- The dimensions of the mip maps are first determined by applying the sizing algorithm presented in Non-Power-of-Two Mipmaps above. Then, if necessary, they are padded out to compression block boundaries.

Surface Arrays

Arrays of 1D and 2D surfaces can be treated as a single surface. This section covers the layout of these composite surfaces.

For All Surfaces

Both 1D and 2D surfaces can be specified as an array. An array surface is indicated by enabling the **Surface Array** field in SURFACE_STATE. 2D multisampled surfaces with Multisampled **Surface Storage Format** set to MSFMT_MSS also are stored like an array in memory.

A value QPitch is defined which indicates the worst-case height for one slice in the texture array. This QPitch is multiplied by the array index to and added to the vertical component of the address to determine the vertical component of the address for that slice. Within the slice, the map is stored identically to a 2D surface.

Since cube surfaces are stored identically to 2D arrays, QPitch is used to determine the spacing between faces of the cube.

For CMS/UMS multisampled surfaces, QPitch is used to determine the spacing between sample slices. For IMS multisampled surfaces, QPitch must account for the additional slice size due to sample storage.

For surfaces defined with SURFACE_STATE, The QPitch field in this state defines the value of QPitch. Software must ensure that QPitch is sufficiently large to avoid overlap between array slices in the memory layout. If an auxiliary surface is defined, a separate QPitch must be set for that surface.

For *depth* and stencil surfaces, QPitch is set in the corresponding state command.

Multisampled Surfaces

Multisampled render targets and sampling engine surfaces are supported. There are three types of multisampled surface layouts designated as follows:

- **IMS** Interleaved Multisampled Surface
- **CMS** Compressed Mulitsampled Surface
- **UMS** Uncompressed Multisampled Surface

These surface layouts are described in the following sections.

Compressed Multisampled Surfaces

Multisampled render targets can be compressed. If **MCS Enable** is enabled in SURFACE_STATE, hardware handles the compression using a software-invisible algorithm. However, performance optimizations in the multisample resolve kernel using the sampling engine are possible if the internal format of these surfaces is understood by software. This section documents the formats of the Multisample Control Surface (MCS) and Multisample Surface (MSS).

The MCS surface consists of one element per pixel, with the element size being an 8-bit unsigned integer value for 4x multisampled surfaces and a 32-bit unsigned integer value for 8x multisampled surfaces. Each field within the element indicates which sample slice (SS) the sample resides on.

For CHV, BSW, the 2x MCS is 8 bits per pixel. The 8 bits are encoded as follows:

2x MCS [CHV, BSW]

7:2	1	0
reserved	sample 1 SS	sample 0 SS

Each 1-bit field indicates which sample slice (SS) the sample's color value is stored. An MCS value of 0x00 indicates that both samples are stored in sample slice 0 (thus have the same color). This is the fully compressed case. An MCS value of 0x03 indicates that all samples in the pixel are in the clear state and none of the sample slices are valid. The pixel's color must be replaced with the surface's clear value.

For CHV, BSW, the 4x MCS is 8 bits per pixel. The 8 bits are encoded as follows:

4x MCS [CHV, BSW]

7:6	5:4	3:2	1:0
sample 3 SS	sample 2 SS	sample 1 SS	sample 0 SS

Each 2-bit field indicates which sample slice (SS) the sample's color value is stored. An MCS value of 0x00 indicates that all four samples are stored in sample slice 0 (thus all have the same color). This is the fully compressed case. An MCS value of 0xff indicates that all samples in the pixel are in the clear state, and none of the sample slices are valid. The pixel's color must be replaced with the surface's clear value.

For CHV, BSW, extending the mechanism used for the 4x MCS to 8x requires 3 bits per sample times 8 samples, or 24 bits per pixel. The 24-bit MCS value per pixel is placed in a 32-bit footprint, with the upper 8 bits unused as shown below.

8x MCS [CHV, BSW]

31:24	23:21	20:18	17:15	14:12	11:9	8:6	5:3	2:0
reserved	sample 7 SS	sample 6 SS	sample 5 SS	sample 4 SS	sample 3 SS	sample 2 SS	sample 1 SS	sample 0 SS

Other than this, the 8x algorithm is the same as the 4x algorithm. The MCS value indicating clear state is 0x00ffffff.

Physical MSS Surface

The physical MSS surface is stored identically to a 2D array surface, with the height and width matching the *pixel* dimensions of the logical multisampled surface. The number of array slices in the physical surface is 2, 4, 8, or 16 times that of the logical surface (depending on the number of multisamples). Sample slices belonging to the same logical surface array slice are stored in adjacent physical slices. The sampling engine *ld2dss* message gives direct access to a specific sample slice.

Uncompressed Multisampled Surfaces

UMS surfaces similar to CMS, except that the MCS is disabled, and there is no MCS surface. UMS contains only an MSS surface, where each sample is stored on its sample slice (SS) of the same index.

Cube Surfaces

The 3D Pipeline supports *cubic environment maps*, conceptually arranged as a cube surrounding the origin of a 3D coordinate system aligned to the cube faces. These maps can be used to supply texel (color/alpha) data of the environment in any direction from the enclosed origin, where the direction is supplied as a 3D "vector" texture coordinate. These cube maps can also be mipmapped.

Each texture map level is represented as a group of six, square *cube face* texture surfaces. The faces are identified by their relationship to the 3D texture coordinate system. The subsections below describe the cube maps as described at the API as well as the memory layout dictated by the hardware.

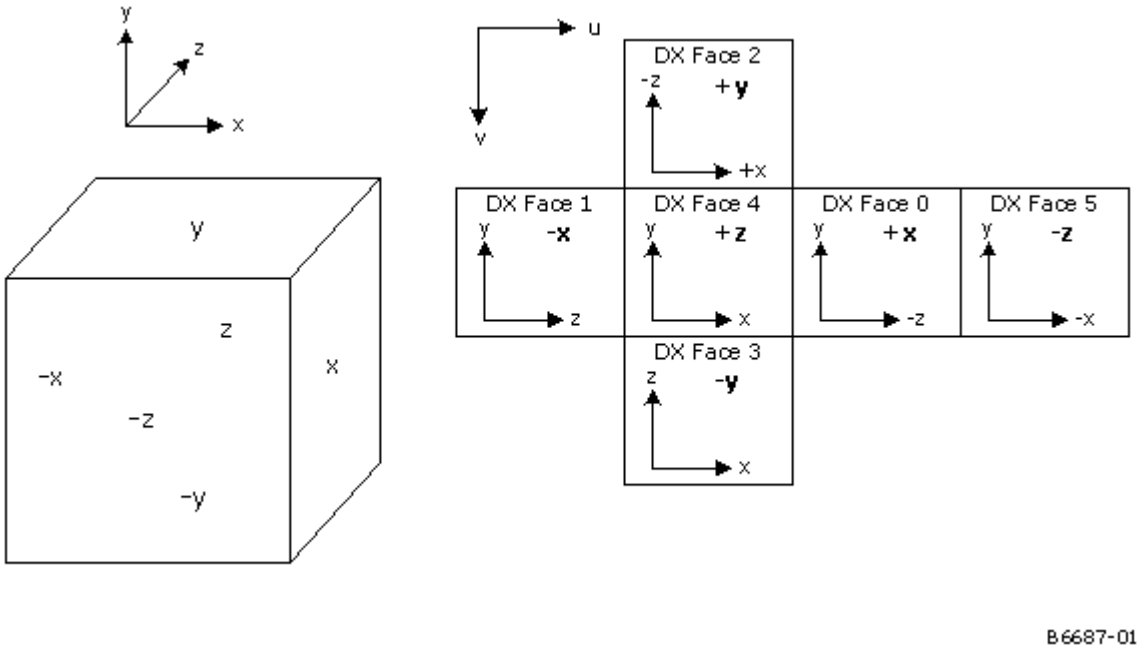
DirectX API Definition

The diagram below describes the cube map faces as they are defined at the DirectX API. It shows the axes on the faces as they would be seen from the inside (at the origin).

The origin of the U,V texel grid is at the top left corner of each face.

This will be looking directly at face 4, the +z -face. Y is up by default.

DirectX Cube Map Definition



Hardware Cube Map Layout

The cube face textures are stored in the same way as 2D array surfaces are stored (see section *2D Surfaces*for details). For cube surfaces, the depth (array instances) is equal to 6. The array index "q" corresponds to the face according to the following table:

"q" coordinate	face
0	+x
1	-x
2	+y
3	-y
4	+z
5	-z

Restrictions

- The cube map memory layout is the same whether or not the cube map is mip-mapped, and whether or not all six faces are "enabled", though the memory backing disabled faces or non-supplied levels can be used by software for other purposes.
- The cube map faces all share the same **Surface Format**

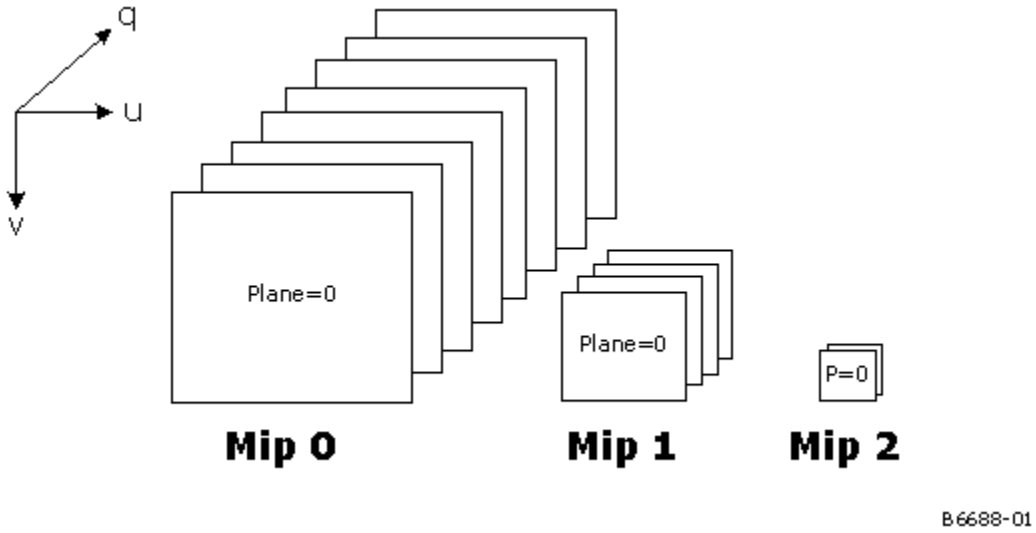
Cube Arrays

Cube arrays are stored identically to 2D surface arrays. A group of 6 consecutive array elements makes up a single cube map. A cube array with N array elements is stored identically to a 2D array with 6N array elements.

3D Surfaces

Multiple texture map surfaces (and their respective mipmap chains) can be arranged into a structure known as a Texture3D (volume) texture. A volume texture map consists of many *planes* of 2D texture maps. See *Sampler* for a description of how volume textures are used.

Volume Texture Map



The number of planes defined at each successive mip level is halved. Volumetric texture maps are stored as follows. All of the LOD=0 q-planes are stacked vertically, then below that, the LOD=1 q-planes are stacked two-wide, then the LOD=2 q-planes are stacked four-wide below that, and so on.

The width, height, and depth of LOD "L" are as follows:

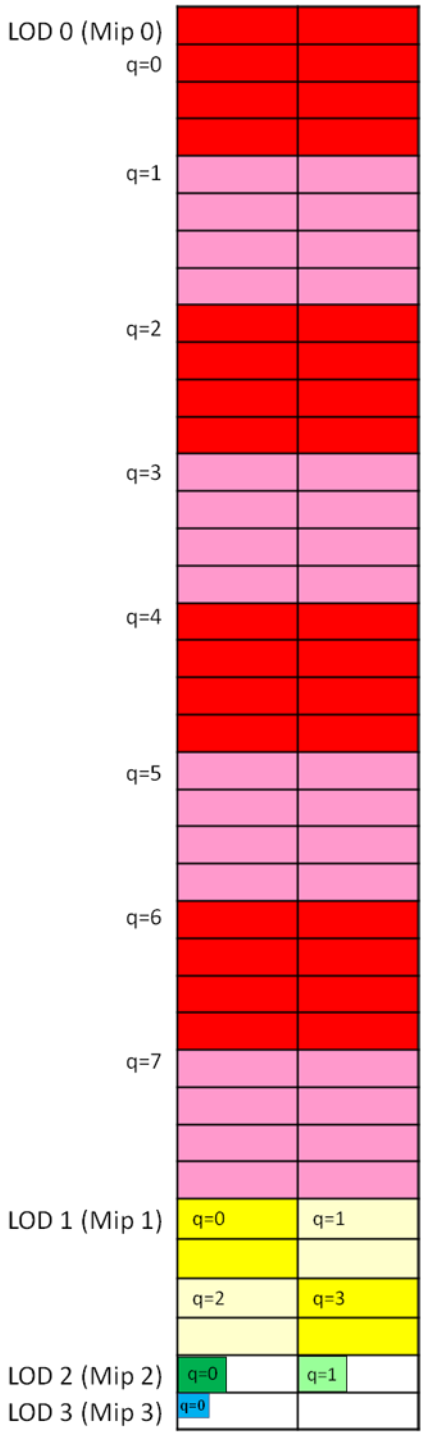
$W_L = ((width » L) > 0 ? width » L:1)$

$H_L = ((height » L) > 0 ? height » L:1)$

This is the same as for a regular texture. For volume textures we add:

$D_L = ((depth » L) > 0 ? depth » L:1)$

Cache-line aligned width and height are as follows, with i and j being a function of the map format as shown in Alignment Unit Size.


LOD 0 (Mip 0)
q=0
q=1
q=2
q=3
q=4
q=5
q=6
q=7
LOD 1 (Mip 1)
q=0
q=1
q=2
q=3
LOD 2 (Mip 2)
q=0
q=1
LOD 3 (Mip 3)
q=0

$$w_L = i * ceil\left(\frac{W_L}{i}\right)$$

$$h_L = j * ceil\left(\frac{H_L}{j}\right)$$

It is not necessary to cache-line align in the "depth" dimension (i.e. lowercase "*d*").

The following equations for $LOD_{L,q}$ give the base address Cartesian coordinates for the map at LOD L and depth q.

$$LOD_{0,q} = (0, q * h_0)$$

$$LOD_{1,q} = ((q\%2) * w_1, D_0 * h_0 + (q >> 1) * h_1)$$

$$LOD_{2,q} = ((q\%4) * w_2, D_0 * h_0 + ceil\left(\frac{D_1}{2}\right) * h_1 + (q >> 2) * h_2)$$

$$LOD_{3,q} = ((q\%8) * w_3, D_0 * h_0 + ceil\left(\frac{D_1}{2}\right) * h_1 + ceil\left(\frac{D_2}{4}\right) * h_2 + (q >> 3) * h_3)$$

These values are then used as "base addresses" and the 2D MIP Map equations are used to compute the location within each LOD/q map.

Minimum Pitch

The minimum pitch required to store the 3D map may in some cases be greater than the minimum pitch required by the LOD=0 map. This is due to cache line alignment requirements that may impact some of the MIP levels requiring additional spacing in the horizontal direction.

Surface Padding Requirements

This section covers the requirements for padding around surfaces stored in memory, as there are cases where the device will overfetch beyond the bounds of the surface due to implementation of caches and other hardware structures.

Alignment Unit Size

This section documents the alignment parameters *i* and *j* that are used depending on the surface.

Alignment Parameters, CHV, BSW

Surface Defined By	Surface Format	Alignment Unit Width "*i*"	Alignment Unit Height "*j*"
3DSTATE_DEPTH_BUFFER	D16_UNORM	8	4
	not D16_UNORM	4	4
3DSTATE_STENCIL_BUFFER	N/A	8	8
SURFACE_STATE	BC*, ETC*, EAC*	4	4
	FXT1	8	4
	ASTC	Value of ASTC_2DBlockWidth (4, 5, 6, 8, 10, or 12)	Value of ASTC_2DBlockHeight (4, 5, 6, 8, 10, or 12)
	all others	set by **Surface Horizontal Alignment**	set by **Surface Vertical Alignment**

Sampling Engine Surfaces

The sampling engine accesses texels outside of the surface if they are contained in the same cache line as texels that are within the surface. These texels will not participate in any calculation performed by the sampling engine and will not affect the result of any sampling engine operation, however if these texels lie outside of defined pages in the GTT, a GTT error will result when the cache line is accessed. In order to avoid these GTT errors, "padding" at the bottom and right side of a sampling engine surface is sometimes necessary.

It is possible that a cache line will straddle a page boundary if the base address or pitch is not aligned. All pages included in the cache lines that are part of the surface must map to valid GTT entries to avoid errors. To determine the necessary padding on the bottom and right side of the surface, refer to the table in Alignment Unit Size section for the i and j parameters for the surface format in use. The surface must then be extended to the next multiple of the alignment unit size in each dimension, and all texels contained in this extended surface must have valid GTT entries.

For example, suppose the surface size is 15 texels by 10 texels and the alignment parameters are i=4 and j=2. In this case, the extended surface would be 16 by 10. Note that these calculations are done in texels, and must be converted to bytes based on the surface format being used to determine whether additional pages need to be defined.

Buffer Padding Requirements

For compressed textures (BC*, FXT1, ETC*, EAC*, and ASTC* surface formats), padding at the bottom of the surface is to an even compressed row. This is equivalent to a multiple of 2*q*, where *q* is the compression block height in texels. Thus, for padding purposes, these surfaces behave as if $j = 2q$ only for surface padding purposes. The value of *j* is still equal to *q* for mip level alignment and QPitch calculation. For cube surfaces, an additional two rows of padding are required at the bottom of the surface. This must be ensured regardless of whether the surface is stored tiled or linear. This is due to the potential rotation of cache line orientation from memory to cache.

For packed YUV, 96 bpt, 48 bpt, and 24 bpt surface formats, additional padding is required. These surfaces require an extra row plus 16 bytes of padding at the bottom in addition to the general padding requirements.

For linear surfaces, additional padding of 64 bytes is required at the bottom of the surface. This is in addition to the padding required above.

Programming Note	
Context:	Sampling Engine Surfaces.
For SURFTYPE_BUFFER, SURFTYPE_1D, and SURFTYPE_2D non-array, non-MSAA, non-mip-mapped surfaces in linear memory, the only padding requirement is to the next aligned 64-byte boundary beyond the end of the surface. The rest of the padding requirements documented above do not apply to these surfaces.	

Programming Note	
Context:	Sampling Engine Surfaces.
For SURFTYPE_2D and SURFTYPE_3D with linear mode and height % 4 != 0, the surface must be padded with 4-(height % 4)***Surface Pitch** # of bytes.	

Programming Note	
Context:	Sampling Engine Surfaces
For SURFTYPE_1D with linear mode, the surface must be padded to 4 times the Surface Pitch # of bytes	

Render Target and Media Surfaces

The data port accesses data (pixels) outside of the surface if they are contained in the same cache request as pixels that are within the surface. These pixels will not be returned by the requesting message, however if these pixels lie outside of defined pages in the GTT, a GTT error will result when the cache request is processed. In order to avoid these GTT errors, “padding” at the bottom of the surface is sometimes necessary.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 6: Command Stream Programming

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Graphics Command Formats **1**
Command Header 2
Memory Interface Commands 6
2D Commands 8
3D Commands 10
VEBOX Commands 14
MFX Commands 15
Scheduling **17**
Execlists 17
Context Descriptor Format 19
Logical Ring Context Format 20
Context Status 21
RINGBUF — Ring Buffer Registers 25
Virtual Memory Control **25**
Render Engine Command Streamer (RCS) **26**
Batch Buffer Privilege Register 27
Mode Registers 27
Logical Context Support 28
Context Save Registers 29
MI Commands for Render Engine 30
User Mode Privileged Commands 31
Watchdog Timer Registers 34
Command Stream Front End **35**
Interrupt Control Registers **35**
Hardware-Detected Error Bit Definitions (for EIR EMR ESR) 37
Blitter Engine Command Streamer (BCS) **38**
Watchdog Timer Registers 39
Hardware-Detected Error Bit Definitions (for EIR EMR ESR) 39
Logical Context Support 40
Mode Registers 41
MI Commands for Blitter Engine 42

User Mode Privileged Commands 45
Video Command Streamer (VCS) 46
Watchdog Timer Registers 47
VCS Hardware - Detected Error Bit Definitions (for EIR EMR ESR) 47
Logical Context Support 48
Mode Registers 49
Registers in Media Engine 49
GFX Pending TLB Cycles Information Registers 49
Memory Interface Commands for Video Codec Engine 50
User Mode Privileged Commands 51
Video Enhancement Engine Command Interface 53
VECS_RINGBUF — Ring Buffer Registers 53
Watchdog Timer Registers 53
Logical Context Support 54
Mode Registers 55
MI Commands for Video Enhancement Engine 56
User Mode Privileged Commands 57
Preemption 58
Ring Buffer Scheduling 58
ExecList Scheduling 60
Command Streamer (CS) ALU Programming 61

Graphics Command Formats

This section describes the general format of the graphics device commands.

Graphics commands are defined with various formats. The first DWord of all commands is called the *header* DWord. The header contains the only field common to all commands, the *client* field that determines the device unit that processes the command data. The Command Parser examines the client field of each command to condition the further processing of the command and route the command data accordingly.

Graphics commands vary in length, though are always multiples of DWords. The length of a command is either:

- Implied by the client/opcode
- Fixed by the client/opcode yet included in a header field (so the Command Parser explicitly knows how much data to copy/process)
- Variable, with a field in the header indicating the total length of the command

Note that command *sequences* require QWord alignment and padding to QWord length to be placed in Ring and Batch Buffers.

The following subsections provide a brief overview of the graphics commands by client type provides a diagram of the formats of the header DWords for all commands. Following that is a list of command mnemonics by client type.

Command Header

Render Command Header Format

Bits						
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**	
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count		
Reserved	001, 010	Opcode – 11111	23:19 Sub Opcode 00h – 01h		18:16 Reserved	15:0 DWord Count

TYPE	**31:29**	**28:27**	**26:24**	**23:16**	**15:8**	**7:0**
Common	011	00	Opcode – 000	Sub Opcode	Data	DWord Count
Common (NP)[1]	011	00	Opcode – 001	Sub Opcode	Data	DWord Count
Reserved	011	00	Opcode – 010 – 111			
Single Dword Command	011	01	Opcode – 000 – 001	Sub Opcode		N/A
Reserved	011	01	Opcode – 010 – 111			
Media State	011	10	Opcode – 000	Sub Opcode		Dword Count
Media Object	011	10	Opcode – 001 – 010	Sub Opcode	Dword Count	
Reserved	011	10	Opcode – 011 – 111			
3DState	011	11	Opcode – 000	Sub Opcode	Data	DWord Count
3DState (NP)[1]	011	11	Opcode – 001	Sub Opcode	Data	DWord Count
PIPE_Control	011	11	Opcode – 010		Data	DWord Count

Bits						
TYPE	31:29	28:24		23	22	21:0
3DPrimitive	011	11	Opcode – 011			Data DWord Count
Reserved	011	11	Opcode – 100 – 111			
Reserved	100	XX				
Reserved	101	XX				
Reserved	110	XX				

Notes:

[1]The qualifier "NP" indicates that the state variable is non-pipelined and the render pipe is flushed before such a state variable is updated. The other state variables are pipelined (default).

Video Command Header Format

Bits						
TYPE	31:29	28:24		23	22	21:0
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Reserved 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds			Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
TYPE	31:29	28:27	26:24	23:16		15:0
Reserved	011	00	XXX	XX		
MFX Single DW	011	01	000	Opcode: 0h		0
Reserved	011	01	1XX			
Reserved	011	10	0XX			
AVC State	011	10	100	Opcode: 0h – 4h		DWord Count
AVC Object	011	10	100	Opcode: 8h		DWord Count
VC1 State	011	10	101	Opcode: 0h – 4h		DWord Count
VC1 Object	011	10	101	Opcode: 8h		DWord Count
Reserved	011	10	11X			
Reserved	011	11	XXX			

TYPE	31:29	28:27	26:24	23:21	20:16	15:0
MFX Common	011	10	000	000	subopcode	DWord Count
Reserved	011	10	000	001-111	subopcode	DWord Count
AVC Common	011	10	001	000	subopcode	DWord Count
AVC Dec	011	10	001	001	subopcode	DWord Count
AVC Enc	011	10	001	010	subopcode	DWord Count
Reserved	011	10	001	011-111	subopcode	DWord Count
Reserved (for VC1 Common)	011	10	010	000	subopcode	DWord Count
VC1 Dec	011	10	010	001	subopcode	DWord Count
Reserved (for VC1 Enc)	011	10	010	010	subopcode	DWord Count
Reserved	011	10	010	011-111	subopcode	DWord Count
Reserved (MPEG2 Common)	011	10	011	000	subopcode	DWord Count
MPEG2 Dec	011	10	011	001	subopcode	DWord Count
Reserved (for MPEG2 Enc)	011	10	011	010	subopcode	DWord Count
Reserved	011	10	011	011-111	subopcode	DWord Count
Reserved	011	10	100-111	XXX		

Video Enhancement Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds		Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count	
Reserved	001, 010				

TYPE	31:29	28:27	26:24	23:21	20:16	15:12	11:0
VEBOX (Parallel Video Pipe)	011	10: Pipeline 00: Reserved 01: Reserved 11: Reserved	Command Opcode – 100	Sub Opcode A	Sub Opcode B	Reserved	Dword Count

Blitter Command Header Format

Bits					
TYPE	**31:29**	**28:24**	**23**	**22**	**21:0**
Memory Interface (MI)	000	Opcode 00h – NOP 0Xh – Single DWord Commands 1Xh – Two+ DWord Commands 2Xh – Store Data Commands 3Xh – Ring/Batch Buffer Cmds			Identification No./DWord Count Command Dependent Data 5:0 – DWord Count 5:0 – DWord Count 5:0 – DWord Count
Reserved	001, 011				

TYPE	31:29	28:22	21:9	8:0
Blitter (2D)	010	Command Opcode	Command Dependent Data	Dword Count

Memory Interface Commands

Memory Interface (MI) commands are basically those commands which do not require processing by the 2D or 3D Rendering/Mapping engines. The functions performed by these commands include:

- Control of the command stream (e.g., Batch Buffer commands, breakpoints, ARB On/Off, etc.)
- Hardware synchronization (e.g., flush, wait-for-event)
- Software synchronization (e.g., Store DWORD, report head)
- Graphics buffer definition (e.g., Display buffer, Overlay buffer)
- Miscellaneous functions

All of the following commands are defined in *Memory Interface Commands*.

Memory Interface Commands for RCP

Opcode (28:23)	Command	Pipes
1 DWord		
00h	MI_NOOP	All
01h	MI_SET_PREDICATE	Render
02h	MI_USER_INTERRUPT	All
03h	MI_WAIT_FOR_EVENT	All
05h	MI_ARB_CHECK	All
06h	MI_RS_CONTROL	Render
07h	MI_REPORT_HEAD	All
08h	MI_ARB_ON_OFF	All except Blitter
09h	MI_URB_ATOMIC_ALLOC	Render
0Ah	MI_BATCH_BUFFER_END	All
0Bh	MI_SUSPEND_FLUSH	All
0Ch	MI_PREDICATE	Render
0Dh	MI_TOPOLOGY_FILTER	Render
0Fh	MI_RS_CONTEXT	Render
2+ DWord		
10h	Reserved	
15h	Reserved	
16h	MI_SEMAPHORE_MBOX	All
17h	Reserved	
18h	MI_SET_CONTEXT	Render
19h	MI_URB_CLEAR	Render
1Ah	MI_MATH	All
1Bh	MI_SEMAPHORE_SIGNAL	All

Opcode (28:23)	Command	Pipes
1 DWord		
1Ch	MI_SEMAPHORE_WAIT	All
1Eh-1Fh	Reserved	
Store Data		
20h	MI_STORE_DATA_IMM	All
21h	MI_STORE_DATA_INDEX	All
22h	MI_LOAD_REGISTER_IMM	All
23h	MI_UPDATE_GTT	All
24h	MI_STORE_REGISTER_MEM	All
26h	MI_FLUSH_DW	All except Render
27h	MI_CLFLUSH	Render
29h	MI_LOAD_REGISTER_MEM	All
2Ah	MI_LOAD_REGISTER_REG	All
2Bh	MI_RS_STORE_DATA_IMM	Render
2Ch	MI_LOAD_URB_MEM	Render
2Dh	MI_STORE_URB_MEM	Render
2Eh	MI_MEM_TO_MEM	All
2Fh	MI_ATOMIC	All
Ring/Batch Buffer		
30h	Reserved	
31h	MI_BATCH_BUFFER_START	Render
32h-35h	Reserved	
36h	MI_CONDITIONAL_BATCH_BUFFER_END	All
37h-38h	Reserved	
39h-3Fh	Reserved	

2D Commands

The 2D commands include various flavors of BLT operations, along with commands to set up BLT engine state without actually performing a BLT. Most commands are of fixed length, though there are a few commands that include a variable amount of "inline" data at the end of the command.

All the following commands are defined in *Blitter Instructions*.

2D Command Map

Opcode (28:22)	Command
00h	Reserved
01h	XY_SETUP_BLT
02h	Reserved
03h	XY_SETUP_CLIP_BLT
04h-10h	Reserved
11h	XY_SETUP_MONO_PATTERN_SL_BLT
12h-23h	Reserved
24h	XY_PIXEL_BLT
25h	XY_SCANLINES_BLT
26h	XY_TEXT_BLT
27h-30h	Reserved
31h	XY_TEXT_IMMEDIATE_BLT
32h-3Fh	Reserved
40h	COLOR_BLT
41h-42h	Reserved
43h	SRC_COPY_BLT
44h-4Fh	Reserved
50h	XY_COLOR_BLT
51h	XY_PAT_BLT
52h	XY_MONO_PAT_BLT
53h	XY_SRC_COPY_BLT
54h	XY_MONO_SRC_COPY_BLT
55h	XY_FULL_BLT
56h	XY_FULL_MONO_SRC_BLT
57h	XY_FULL_MONO_PATTERN_BLT
58h	XY_FULL_MONO_PATTERN_MONO_SRC_BLT
59h	XY_MONO_PAT_FIXED_BLT
5Ah-70h	Reserved

Opcode (28:22)	Command
71h	XY_MONO_SRC_COPY_IMMEDIATE_BLT
72h	XY_PAT_BLT_IMMEDIATE
73h	XY_SRC_COPY_CHROMA_BLT
74h	XY_FULL_IMMEDIATE_PATTERN_BLT
75h	XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT
76h	XY_PAT_CHROMA_BLT
77h	XY_PAT_CHROMA_BLT_IMMEDIATE
78h-7Fh	Reserved

3D Commands

The 3D commands are used to program the graphics pipelines for 3D operations.

Refer to the *3D* chapter for a description of the 3D state and primitive commands and the *Media* chapter for a description of the media-related state and object commands.

For all commands listed in **3D Command Map**, the Pipeline Type (bits 28:27) is 3h, indicating the 3D Pipeline.

3D Command Map

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	03h	Reserved	
0h	04h	3DSTATE_CLEAR_PARAMS	3D Pipeline
0h	05h	3DSTATE_DEPTH_BUFFER	3D Pipeline
0h	06h	3DSTATE_STENCIL_BUFFER	3D Pipeline
0h	07h	3DSTATE_HIER_DEPTH_BUFFER	3D Pipeline
0h	08h	3DSTATE_VERTEX_BUFFERS	Vertex Fetch
0h	09h	3DSTATE_VERTEX_ELEMENTS	Vertex Fetch
0h	0Ah	3DSTATE_INDEX_BUFFER	Vertex Fetch
0h	0Bh	3DSTATE_VF_STATISTICS	Vertex Fetch
0h	0Ch	Reserved	
0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS	3D Pipeline
0h	0Eh	3DSTATE_CC_STATE_POINTERS	3D Pipeline
0h	10h	3DSTATE_VS	Vertex Shader
0h	11h	3DSTATE_GS	Geometry Shader
0h	12h	3DSTATE_CLIP	Clipper
0h	13h	3DSTATE_SF	Strips & Fans
0h	14h	3DSTATE_WM	Windower
0h	15h	3DSTATE_CONSTANT_VS	Vertex Shader
0h	16h	3DSTATE_CONSTANT_GS	Geometry Shader
0h	17h	3DSTATE_CONSTANT_PS	Windower
0h	18h	3DSTATE_SAMPLE_MASK	Windower
0h	19h	3DSTATE_CONSTANT_HS	Hull Shader
0h	1Ah	3DSTATE_CONSTANT_DS	Domain Shader
0h	1Bh	3DSTATE_HS	Hull Shader
0h	1Ch	3DSTATE_TE	Tesselator
0h	1Dh	3DSTATE_DS	Domain Shader
0h	1Eh	3DSTATE_STREAMOUT	HW Streamout

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	1Fh	3DSTATE_SBE	Setup
0h	20h	3DSTATE_PS	Pixel Shader
0h	21h	3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP	Strips & Fans
0h	22h	Reserved	
0h	23h	3DSTATE_VIEWPORT_STATE_POINTERS_CC	Windower
0h	24h	3DSTATE_BLEND_STATE_POINTERS	Pixel Shader
0h	25h	3DSTATE_DEPTH_STENCIL_STATE_POINTERS	Pixel Shader
0h	26h	3DSTATE_BINDING_TABLE_POINTERS_VS	Vertex Shader
0h	27h	3DSTATE_BINDING_TABLE_POINTERS_HS	Hull Shader
0h	28h	3DSTATE_BINDING_TABLE_POINTERS_DS	Domain Shader
0h	29h	3DSTATE_BINDING_TABLE_POINTERS_GS	Geometry Shader
0h	2Ah	3DSTATE_BINDING_TABLE_POINTERS_PS	Pixel Shader
0h	2Bh	3DSTATE_SAMPLER_STATE_POINTERS_VS	Vertex Shader
0h	2Ch	3DSTATE_SAMPLER_STATE_POINTERS_HS	Hull Shader
0h	2Dh	3DSTATE_SAMPLER_STATE_POINTERS_DS	Domain Shader
0h	2Eh	3DSTATE_SAMPLER_STATE_POINTERS_GS	Geometry Shader
0h	2Fh	Reserved	
0h	30h	3DSTATE_URB_VS	Vertex Shader
0h	31h	3DSTATE_URB_HS	Hull Shader
0h	32h	3DSTATE_URB_DS	Domain Shader
0h	33h	3DSTATE_URB_GS	Geometry Shader
0h	34h	3DSTATE_GATHER_CONSTANT_VS	Vertex Shader
0h	35h	3DSTATE_GATHER_CONSTANT_GS	Geometry Shader
0h	36h	3DSTATE_GATHER_CONSTANT_HS	Hull Shader
0h	37h	3DSTATE_GATHER_CONSTANT_DS	Domain Shader
0h	38h	3DSTATE_GATHER_CONSTANT_PS	Pixel Shader
0h	39h	3DSTATE_DX9_CONSTANTF_VS	Vertex Shader
0h	3Ah	3DSTATE_DX9_CONSTANTF_PS	Pixel Shader
0h	3Bh	3DSTATE_DX9_CONSTANTI_VS	Vertex Shader
0h	3Ch	3DSTATE_DX9_CONSTANTI_PS	Pixel Shader
0h	3Dh	3DSTATE_DX9_CONSTANTB_VS	Vertex Shader
0h	3Eh	3DSTATE_DX9_CONSTANTB_PS	Pixel Shader
0h	3Fh	3DSTATE_DX9_LOCAL_VALID_VS	Vertex Shader
0h	40h	3DSTATE_DX9_LOCAL_VALID_PS	Pixel Shader
0h	41h	3DSTATE_DX9_GENERATE_ACTIVE_VS	Vertex Shader
0h	42h	3DSTATE_DX9_GENERATE_ACTIVE_PS	Pixel Shader

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
0h	43h	3DSTATE_BINDING_TABLE_EDIT_VS	Vertex Shader
0h	44h	3DSTATE_BINDING_TABLE_EDIT_GS	Geometry Shader
0h	45h	3DSTATE_BINDING_TABLE_EDIT_HS	Hull Shader
0h	46h	3DSTATE_BINDING_TABLE_EDIT_DS	Domain Shader
0h	47h	3DSTATE_BINDING_TABLE_EDIT_PS	Pixel Shader
0h	48h	3DSTATE_VF_HASHING	Vertex Fetch
0h	49h	3DSTATE_VF_INSTANCING	Vertex Fetch
0h	4Ah	3DSTATE_VF_SGVS	Vertex Fetch
0h	4Bh	3DSTATE_VF_TOPOLOGY	Vertex Fetch
0h	4Ch	3DSTATE_WM_CHROMA_KEY	Windower
0h	4Dh	3DSTATE_PS_BLEND	Windower
0h	4Eh	3DSTATE_WM_DEPTH_STENCIL	Windower
0h	4Fh	3DSTATE_PS_EXTRA	Windower
0h	50h	3DSTATE_RASTER	Strips & Fans
0h	51h	3DSTATE_SBE_SWIZ	Strips & Fans
0h	52h	3DSTATE_WM_HZ_OP	Windower
0h	53h	3DSTATE_INT (internally generated state)	3D Pipeline
0h	56h	Reserved	
0h	57h-59h	Reserved	
	60h-68h	Reserved	
	69h	Reserved	
0h	6Ah-6Bh	Reserved	
0h	6Ch-FFh	Reserved	
1h	00h	3DSTATE_DRAWING_RECTANGLE	Strips & Fans
1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0	Sampling Engine
1h	03h	Reserved	
1h	04h	3DSTATE_CHROMA_KEY	Sampling Engine
1h	05h	Reserved	
1h	06h	3DSTATE_POLY_STIPPLE_OFFSET	Windower
1h	07h	3DSTATE_POLY_STIPPLE_PATTERN	Windower
1h	08h	3DSTATE_LINE_STIPPLE	Windower
1h	0Ah	3DSTATE_AA_LINE_PARAMS	Windower
1h	0Bh	3DSTATE_GS_SVB_INDEX	Geometry Shader
1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1	Sampling Engine
1h	0Dh	3DSTATE_MULTISAMPLE	Windower
1h	0Eh	3DSTATE_STENCIL_BUFFER	Windower

Opcode Bits 26:24	Sub Opcode Bits 23:16	Command	Definition Chapter
1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER	Windower
1h	10h	3DSTATE_CLEAR_PARAMS	Windower
1h	11h	3DSTATE_MONOFILTER_SIZE	Sampling Engine
1h	12h	3DSTATE_PUSH_CONSTANT_ALLOC_VS	Vertex Shader
1h	13h	3DSTATE_PUSH_CONSTANT_ALLOC_HS	Hull Shader
1h	14h	3DSTATE_PUSH_CONSTANT_ALLOC_DS	Domain Shader
1h	15h	3DSTATE_PUSH_CONSTANT_ALLOC_GS	Geometry Shader
1h	16h	3DSTATE_PUSH_CONSTANT_ALLOC_PS	Pixel Shader
1h	17h	3DSTATE_SO_DECL_LIST	HW Streamout
1h	18h	3DSTATE_SO_BUFFER	HW Streamout
1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC	Resource Streamer
1h	1Ah	3DSTATE_GATHER_POOL_ALLOC	Resource Streamer
1h	1Bh	3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC	Resource Streamer
1h	1Ch	3DSTATE_SAMPLE_PATTERN	Windower
1h	1Dh	3DSTATE_URB_CLEAR	3D Pipeline
1h	1Eh-FFh	Reserved	
2h	00h	PIPE_CONTROL	3D Pipeline
2h	01h-FFh	Reserved	
3h	00h	3DPRIMITIVE	Vertex Fetch
3h	01h-FFh	Reserved	
4h-7h	00h-FFh	Reserved	

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
Common (pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	0h	03h	STATE_PREFETCH	Graphics Processing Engine
0h	0h	04h-FFh	Reserved	
Common (non-pipelined)	**Bits 26:24**	**Bits 23:16**		
0h	1h	00h	Reserved	N/A
0h	1h	01h	STATE_BASE_ADDRESS	Graphics Processing Engine
0h	1h	02h	STATE_SIP	Graphics Processing Engine
0h	1h	03h	SWTESS BASE ADDRESS	3D Pipeline
0h	1h	04h	GPGPU CSR BASE ADDRESS	Graphics Processing Engine
0h	1h	05h-1Dh	Reserved	
0h	1h	1Eh	Reserved	
0h	1h	1Fh-20h	Reserved	
0h	1h	21h-24h	Reserved	

Pipeline Type (28:27)	Opcode	Sub Opcode	Command	Definition Chapter
Common (pipelined)	Bits 26:24	Bits 23:16		
0h	1h	25h–FFh	Reserved	N/A
Reserved	Bits 26:24	Bits 23:16		
0h	2h–7h	XX	Reserved	N/A

VEBOX Commands

The VEBOX commands are used to program the Video Enhancement engine attached to the Video Enhancement Command Parser.

VEBOX Command Map

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command
2h	4h	0h	0h	VEBOX_SURFACE_STATE
2h	4h	0h	2h	VEBOX_STATE

MFX Commands

The MFX (MFD for decode and MFC for encode) commands are used to program the multi-format codec engine attached to the Video Codec Command Parser. See the *MFD* and *MFC* chapters for a description of these commands.

MFX state commands support direct state model and indirect state model. Recommended usage of indirect state model is provided here (as a software usage guideline).

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
MFX Common (State)							
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	N/A
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	N/A
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	N/A
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	N/A
2h	0h	0h	7-8h	Reserved	N/A	N/A	N/A
MFX Common (Object)							
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	N/A	Yes
2h	0h	0h	4-1Fh	Reserved	N/A	N/A	N/A
AVC Common (State)							
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	N/A
2h	1h	0h	1h	MFX_AVC_QM_STATE	MFX	IMAGE	N/A
2h	1h	0h	2h	MFX_AVC_DIRECTMODE_STATE	MFX	SLICE	N/A
2h	1h	0h	3h	MFX_AVC_SLICE_STATE	MFX	SLICE	N/A
2h	1h	0h	4h	MFX_AVC_REF_IDX_STATE	MFX	SLICE	N/A
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	N/A
2h	1h	0h	6-1Fh	Reserved	N/A	N/A	N/A
AVC Dec							
2h	1h	1h	0-7h	Reserved	N/A	N/A	N/A
2h	1h	1h	8h	MFD_AVC_BSD_OBJECT	MFX	N/A	No
2h	1h	1h	9-1Fh	Reserved	N/A	N/A	N/A
AVC Enc							
2h	1h	2h	0-1h	Reserved	N/A	N/A	N/A
2h	1h	2h	2h	MFC_AVC_FQM_STATE	MFX	IMAGE	N/A
2h	1h	2h	3-7h	Reserved	N/A	N/A	N/A
2h	1h	2h	8h	MFC_AVC_PAK_INSERT_OBJECT	MFX	N/A	N/A

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	N/A	Yes
2h	1h	2h	A-1Fh	Reserved	N/A	N/A	N/A
2h	1h	2h	0-1Fh	Reserved	N/A	N/A	N/A
VC1 Common							
2h	2h	0h	0h	MFX_VC1_PIC_STATE	MFX	IMAGE	N/A
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	N/A
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	N/A
2h	2h	0h	2-1Fh	Reserved	N/A	N/A	N/A
VC1 Dec							
2h	2h	1h	0-7h	Reserved	N/A	N/A	N/A
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	N/A	Yes
2h	2h	1h	9-1Fh	Reserved	N/A	N/A	N/A
VC1 Enc							
2h	2h	2h	0-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Common							
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	N/A
2h	3h	0h	1h	MFX_MPEG2_QM_STATE	MFX	IMAGE	N/A
2h	3h	0h	2-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Dec							
2h	3h	1h	1-7h	Reserved	N/A	N/A	N/A
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	N/A	Yes
2h	3h	1h	9-1Fh	Reserved	N/A	N/A	N/A
MPEG2 Enc							
2h	3h	2h	0-1Fh	Reserved	N/A	N/A	N/A
The Rest							
2h	4-5h, 7h	x	x	Reserved	N/A	N/A	N/A

Scheduling

Content for this topic is currently under development.

Execlists

Execlists are the method by which new contexts are submitted for execution. Note that this mechanism cannot be used when the **Execlist Enable** bit in the corresponding engines MODE register is not set, i.e GFX_MODE register for Render Engine, BLT_MODE register for Blitter Engine, VCS_MODE register for Video Engine, or VECS_MODE register for Video Enhancement Engine. If this bit is not set in the engine's MODE register, writing to the registers in this section is UNDEFINED.

Broadwell implements two execlists. Each execlist can have up to two context descriptors in it, each describing a context to run. SW assembles an execlist by writing each of the context descriptor elements to the Execlist Submit Port register. Writing the final DWord triggers the submission. It is the responsibility of SW to keep track of when an empty execlist entry is available to receive a new execlist submitted via the Submit Port. Submitting a new execlist when there is already a pending execlist (in addition to the current execlist) is UNDEFINED. In general, the interrupt indicating that the pending execlist has become the current execlist should always be observed before a new pending execlist is submitted. This includes the case where the ring is idle and the very first execlist is submitted; it should not be assumed that this execlist becomes the current list instantaneously.

The submission of a new execlist (known as a preemption request) is interpreted as a request to switch execlists as soon as possible. This is the only trigger for a execlist switch. Within an execlist, a switch from one element (context) to the next can be triggered for several reasons, all of which are synchronous to what the running context itself is doing. Once a context is switched out, the relevant context state and context descriptor doesn't exist in HW, only way the context can be brought back in to HW is by SW resubmitting the context through Execlist Submit Port.

SW must ensure the contexts submitted to both the context descriptors in the execlist are different; i.e SW must not submit the same context descriptor to both the elements of the execlist.

The following are Execlist Registers:

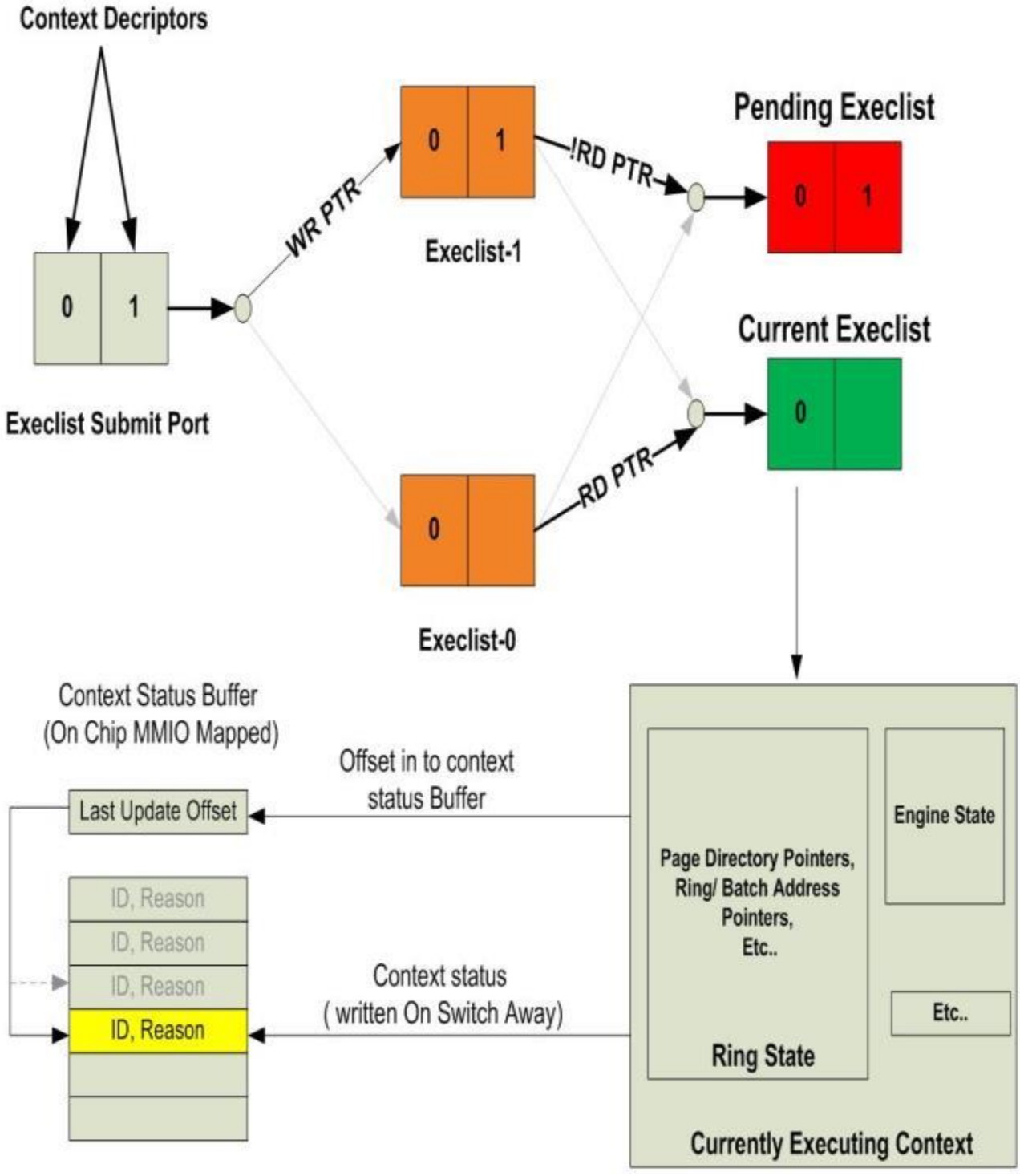
Execlist Submit Port Register

Execlist 0 Contents Register

Execlist 1 Contents

EXECLIST_STATUS - Execlist Status Register

Execlist Structure [CHV]



Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Context Descriptor Format Structure

Context Descriptor Format

Context Descriptor Format

Before submitting a context for the first time, the context image must be properly initialized. Proper initialization includes the ring context registers (ring location, head/tail pointers, etc.) and the page directory.

Render CS Only: Render state need not be initialized; the **Render Context Restore Inhibit** bit in the Context/Save image in memory should be set to prevent restoring garbage render context. See the Logical Ring Context Format section for details.

Programming Note on Context ID field in the Context Descriptor

This section describes the current usage by SW.

General Layout:

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
Eng. ID				**SW Counter**					**HW Use**		**SW Context ID**																					

Eng. ID = Engine ID (a software defined enum to identify RCS, BCS etc..)

SW Counter = Submission Counter. (SW generates an unique counter value on every submission to ensure GroupID + PASID is unique to avoid ambiguity in fault reporting & handling)

Bit 20 = Is Proxy submission. If Set to true, SW Context ID[19:0] = LRCA [31:20], else it is an index into the Context Pool.

Logical Ring Context Format

When execlists are enabled, the Context Image for the each engine (Render, Video, Blitter, Video Enhancement) consists of Per-Process HW Status Page, Ring Context and Engine Context:

- Per-Process HW Status Page (4K)
- Ring Context (Ring Buffer Control Registers, Page Directory Pointers ..etc)
- Engine Context (PipelineState, Non-pipelineState, Statistics)

When execlists are disabled, the context image doesn't consist the Per-Process HW status page.

The detailed format of the Blitter/Video/VideoEnahancement logical ring context is documented in the "GPU Overview" volume, "Memory Data Formats" chapter.

The detailed format of the Render logical Ring and Engine Context including their size is mentioned in section "Graphics BSpec: 3D-Media-GPGPU Engine > Render Command Memory Interface > Render Engine Logical Context Data > Register/State Context" for each product.

Context Status

A context switch interrupt will be sent anytime a context switch or execlist change occurs (including the execlist change without context switch scenario described in the ELSP -- Execlist Submit Port Register section). A status QW for the context that was just switched away from will be written to the Context Status Buffer in the Global Hardware Status Page. A copy of the Context Status Buffer is also maintained ON CHIP inside the command streamer, which is MMIO mapped and can be read/written using MMIO access.

Context Status Buffer in Global Hardware Status Page is exercised when IA based scheduling is done. The status contains the context ID and the reason for the context switch. Note that since there will have been no running contexts when the very first (after reset) execlist is submitted or when HW is idle, the Context ID in the first Context Status Qword will be UNDEFINED, this is indicated by setting IDLE to ACTIVE bit in the context status.

Format of Context Status QWord

Bits	Description
63:32	**Context ID**
31:29	Reserved
28	Reserved
28	**POSH Context Complete** States the state of the POSH pipeline on a context switch by render pipe. When set indicates POSH pipe has completely processed (Head eqv to Tail) the submitted workload.
27:24	Reserved
23:20	Reserved
19:16	Reserved
19:16	**Display Plane.** This indicates the display plane for which Wait on Scanline/V-Blank/Sync Flip has been executed leading to context switch. This field when set to “0000b” indicates display plane on which Wait on Scanline/V-Blank/Sync Flip has been executed is indicated on bits [14:12]. This field is only valid when one of the “Wait on Scanline” or “Wait on Vblnak” or “Wait on sync Flip” is set. 0000b: Reserved (future Sprite A) (Look at field 14:12) 0001b: Reserved (future Sprite B) 0010b: Reserved (future Sprite C) 0011b: Display Plane Sprite A2 0100b: Display Plane Sprite B2 0101b: Display Plane Sprite C2 0110b: Display Plane Sprite A3 0111b: Display Plane Sprite B3 1000b: Display Plane Sprite C3 1001b to 1111b: Reserved

Bits	Description
15	**Lite Restore.** This bit is only valid only when Preempted bit is set. When set, this bit indicates a given context got preempted with the same context resulting in Lite Restore in HW.
14:12	**Display Plane.** This indicates the display plane for which Wait on Scnaline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the “Wait on Scanline” or “Wait on Vblnak” or “Wait on sync Flip” is set. 000b: Display Plane-A 001b: Display Plane-B 010b: Display Plane-C 011b: Display Plane Sprite-A 100b: Display Plane Sprite-B 101b: Display Plane Sprite-C
14:12	**Display Plane.** This indicates the display plane for which Wait on Scnaline/V-Blank/Sync Flip has been executed leading to context switch. This field is only valid when one of the “Wait on Scanline” or “Wait on Vblnak” or “Wait on sync Flip” is set. This field must be looked at only when Display Plane on bits [19:16] is “0000b” and when one of the “Wait on Scanline” or “Wait on Vblank” or “Wait on sync Flip” is set. 000b: Display Plane-A 001b: Display Plane-B 010b: Display Plane-C 011b: Display Plane Sprite-A 100b: Display Plane Sprite-B 101b: Display Plane Sprite-C
11	**Semaphore Wait Mode** 0: Signal Mode 1: Poll Mode This field is valid and must be looked at only when the "Wait on Semaphore" field is set.
10	Reserved
9	Reserved
8	**Wait on Scanline** has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible.
8	**Wait on Scanline** has resulted in context switch.

Bits	Description
7	**Wait on Semaphore** has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible.
7	**Wait on Semaphore** has resulted in context switch.
6	**Wait on V-Blank** has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible.
6	**Wait on V-Blank** has resulted in context switch.
5	**Wait on Sync Flip** has resulted in context switch. This may be cleared during context exit if the condition is met. If none of the bits specifying why we exit and ring is not complete, then scheduler should reschedule the context when possible.
5	**Wait on Sync Flip** has resulted in context switch.
4	**Context Complete** Element is completely processed (Head eqv to Tail) and resulted in a context switch.
3	**ACTIVE to IDLE** following this context switch there is no active element available in HW to execute.
2	**Element Switch.** Context Switch happened from first element in the current execlist to the second element of the same execlist.
1	**Preempted.** Submission of a new execlist has resulted in context switch. The switch is from element in current execlist to element in pending execlist.
0	**IDLE to ACTIVE.** Execlist Submitted when HW is IDLE. When this bit is set rest of the fields in CSQ are not valid.

Context Status should be inferred as described in the tables below. In the two tables below only one of the context switch types will be set and it's quite possible multiple context switch reasons are set. A "Y" in a cell indicates the possibility of the context switch type for the corresponding context switch reason.

Inference of Context Status

Ctx Switch Type Ctx Switch Reason	IDLE to Active	Preempted/ Execlist Switch	**Element Switch	ACTIVE to IDLE
Context Complete	X	Y	Y	Y
Wait on Sync Flip	X	Y	Y	Y
Wait on V-Blank	X	Y	Y	Y
Wait on ScanLine	X	Y	Y	Y
Wait on Semaphore	X	Y	Y	Y
High Priority Context	Y	Y	X	Y

** This field is not valid when High Priority Context field is set and HW must force it to '0'.

When SW services a context switch interrupt, it should read the Context Status Buffer beginning where it left off reading the last time it serviced a context switch interrupt. It should read up to the **Context Status Buffer Write Pointer,** which is recorded in the Context Status Buffer Pointer register. At the end of the context switch interrupt processing SW will update the **Context Status Buffer Read Pointer** with the write buffer pointer value. The status QWs can be examined to determine which contexts were switched out between context interrupt service intervals, and why.

Number of Context Status Entries

Project	Number of Status Entries
CHV	6 (QW) Entries

Status QWords are written to the Context Status Buffer at incrementing locations. The Context Status Buffer has a limited size (see Table Number of Context Status Entries) and simply wraps around to the beginning when the end is reached. Normally the number of status updates that can occur without SW intervening to submit a new execlist (and presumably reading any new status) is the number of execlists times the maximum number of context elements per execlist. Also note that there is no predictable relationship between a context's position in an execlist and the position of its corresponding status QWord in the Context Status Buffer.

The Context Status Buffer fits into a single cacheline so that the whole buffer is read from memory at once if the driver performs a cacheable read.

Format of the Context Status Buffer

QW	Description
7	**Last Written Status Offset.** The lower byte of this QWord is written on every context switch with the (pre-increment) value of the b>Context Status Buffer Write Pointer. The lower 3 bits increment for every status QWord write; bits[7:3] are reserved and must be '0'. The lowest 3 bits indicate which of the Context Status QWords was just written. The rest of the bits [63:8] are reserved.
6	Reserved: MBZ
5:0	**Context Status QWords.** A circular buffer of context status QWs. As each context is switched away from, its status is written here at ascending QWs as indicated by the **Last Written Status Offset**. Once QW 5 has been written, the pointer wraps around so that the next status will be written at QW0. Format = ContextStatusDW

The following are Context Status Registers:

CTXT_ST_PTR - Context Status Buffer Pointer Register

CTXT_ST_BUF - Context Status Buffer Contents

RINGBUF — Ring Buffer Registers

See the "Device Programming Environment" chapter for detailed information on these registers.

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Virtual Memory Control

Per-Process GTT (PPGTT) is setup for an engine (Render, Blitter, Video and Video Enhancement) by programming corresponding Page Directory Pointer (PDP) registers listed below. Refer "Graphics Translation Tables" in "Memory Overview" for more details on Per-Process page table entries and related translations.

Render Engine Command Streamer (RCS)

The RCS (Render Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Render Engine. It is responsible for fetching, decoding, and dispatching of data packets (3D/Media Commands with the header DWord removed) to the front end interface module of Render Engine.

Logic Functions Included
• MMIO register programming interface. • DMA action for fetching of ring data from memory. • Management of the Head pointer for the Ring Buffer. • Decode of ring data and sending it to the appropriate destination: 3D (Vertex Fetch Unit) & GPGPU. • Handling of user interrupts. • Flushing the 3D and GPGPU Engine. • Handle NOP.
• DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

The register programming bus is a DWord interface bus that is driven by the configuration master. The RCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x2000 to 0x27FF. The Gx and MFX Engines use semaphore to synchronize their operations.

RCS operates completely independent of the MFx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside RCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (8 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards Vertex Fetch Unit or GPPGU engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Batch Buffer Privilege Register

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

Mode Registers

The following are the Mode Registers:

INSTPM - Instruction Parser Mode Register

EXCC - Execute Condition Code Register

NOPID - NOP Identification Register

CSPREEMPT - CSPREEMPT

RCS_CTXID_PREEMPTION_HINT - RCS Context Preemption Hint

RCS_PREEMPTION_HINT - RCS_PREEMPTION_HINT

RCS_PREEMPTION_HINT_UDW - RCS_PREEMPTION_HINT_UDW

RS_PRE_HINT - RS Preemption Hint

RS_PREEMPTION_HINT_UDW - RS Preemption Hint UDW

IDLEDLY - Idle Switch Delay

SEMA_WAIT_POLL - Semaphore Polling Interval on Wait

CS_RESET_CTRL - CS Reset Control Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

Logical Context Support

The following are the Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

RCS_BB_STATE - RCS Batch Buffer State Register

CTXT_SR_CTL - Context Save/Restore Control Register

CCID - Current Context Register

CXT_SIZE - Context Sizes

RS_CXT_OFFSET - Resource Streamer Context Offset

URB_CXT_OFFSET - URB Context Offset

CXT_EL_OFFSET - Exec-List Context Offset

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

WAIT_FOR_RC6_EXIT - Control Register for Power Management

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register

SBB_STATE - Second Level Batch Buffer State Register

PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0

PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1

PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2

PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0

PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1

PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

R_PWR_CLK_STATE - Render Power Clock State Register

Context Save Registers

The following are the Context Save Registers:

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

BB_START_ADDR - Batch Buffer Start Head Pointer Register

BB_START_ADDR_UDW - Batch Buffer Start Head Pointer Register for Upper DWord

BB_ADDR_DIFF - Batch Address Difference Register

BB_OFFSET - Batch Offset Register

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

TDL_CONTEXT_SAVE - Context Save Request to TDL

MI Commands for Render Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the original graphics processing engine. The term "for Rendering Engine" in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

MI_NOOP

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_START

MI_CONDITIONAL_BATCH_BUFFER_END

MI_DISPLAY_FLIP

MI_DISPLAY_FLIP

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_INCL

MI_CLFLUSH

MI_MATH

MI_REPORT_HEAD

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_ATOMIC

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_MEM

MI_STORE_REGISTER_MEM

MI_SUSPEND_FLUSH

MI_UPDATE_GTT

MI_USER_INTERRUPT

MI_WAIT_FOR_EVENT

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_WAIT

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring buffer. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as "Command Parser Master Error" interrupt to SW. Privilege access violation checks in HW can be disabled by setting "Privilege Check Disable" bit in GFX_MODE register. When privilege access checks are disabled HW will execute the Privilege command as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
MI_UPDATE_GTT	Command is converted to NOOP.
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_STORE_DATA_INDEX	Command is converted to NOOP.
MI_STORE_REGISTER_MEM	Register read is always performed. Memory update is dropped if **Use Global GTT** is enabled.
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Command is converted to NOOP, if the register accessed is privileged.
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** will be discarded.
MI_REPORT_PERF_COUNT	Command is converted to NOOP if **Use Global GTT** is enabled If a privilege access violation occurs, the Report ID field in the report generated by the next legitimate MI_REPORT_PERF_COUNT may be corrupted.
PIPE_CONTROL	Still send flush down, Post-Sync Operation is NOOP if **Use Global GTT** or Use "Store Data Index" is enabled. Post-Sync Operation LRI to **Privileged Register** is discarded.

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
	Programming Note **Context:** User Mode Privileged Commands Command Privilege Violation error will not be set in a predictable fashion on a Privilege Access Violation from a PIPE_CONTROL command.
MI_SET_CONTEXT	Command is converted to NOOP.
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_ARB_ON_OFF	Command is converted to NOOP.
MI_DISPLAY_FLIP	Command is converted to NOOP.
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.

Parsing one of the commands in the table above from a non-privileged batch buffer will flag an error and convert the command to a NOOP.

The table below lists the non-privileged registers that can be written to from a non-privileged batch buffer executed from Render Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	SIZE in DWords
NOPID	0x2094	1
MI_PREDICATE_RESULT_2	0x23BC	1
INSTPM	0x20C0	1
IA_VERTICES_COUNT	0x2310	2
IA_PRIMITIVES_COUNT	0x2318	2
VS_INVOCATION_COUNT	0x2320	2
HS_INVOCATION_COUNT	0x2300	2
DS_INVOCATION_COUNT	0x2308	2
GS_INVOCATION_COUNT	0x2328	2
GS_PRIMITIVES_COUNT	0x2330	2
CL_INVOCATION_COUNT	0x2338	2
CL_PRIMITIVES_COUNT	0x2340	2
PS_INVOCATION_COUNT_0	0x22C8	2
PS_DEPTH_COUNT _0	0x22D8	2
GPUGPU_DISPATCHDIMX	0x2500	1
GPUGPU_DISPATCHDIMY	0x2504	1
GPUGPU_DISPATCHDIMZ	0x2508	1

MMIO Name	MMIO Offset	SIZE in DWords
MI_PREDICATE_SRC0	0x2400	1
MI_PREDICATE_SRC0	0x2404	1
MI_PREDICATE_SRC1	0x2408	1
MI_PREDICATE_SRC1	0x240C	1
MI_PREDICATE_DATA	0x2410	1
MI_PREDICATE_DATA	0x2414	1
MI_PRED_RESULT	0x2418	1
3DPRIM_END_OFFSET	0x2420	1
3DPRIM_START_VERTEX	0x2430	1
3DPRIM_VERTEX_COUNT	0x2434	1
3DPRIM_INSTANCE_COUNT	0x2438	1
3DPRIM_START_INSTANCE	0x243C	1
3DPRIM_BASE_VERTEX	0x2440	1
GPGPU_THREADS_DISPATCHED	0x2290	2
PS_INVOCATION_COUNT_1	0x22F0	2
PS_DEPTH_COUNT _1	0x22F8	2
BB_OFFSET	0x2158	1
MI_PREDICATE_RESULT_1	0x241C	1
CS_GPR (1-16)	0x2600	32
PS_INVOCATION_COUNT_2	0x2448	2
PS_DEPTH_COUNT_2	0x2450	2
Cache_Mode_0	0x7000	1
Cache_Mode_1	0x7004	1
GT_MODE	0x7008	1
L3_Config	0x7034	1
TD_CTL	0xE400	1
TD_CTL2	0xE404	1
SO_NUM_PRIMS_WRITTEN0	0x5200	2
SO_NUM_PRIMS_WRITTEN1	0x5208	2
SO_NUM_PRIMS_WRITTEN2	0x5210	2
SO_NUM_PRIMS_WRITTEN3	0x5218	2
SO_PRIM_STORAGE_NEEDED0	0x5240	2
SO_PRIM_STORAGE_NEEDED1	0x5248	2
SO_PRIM_STORAGE_NEEDED2	0x5250	2
SO_PRIM_STORAGE_NEEDED3	0x5258	2
SO_WRITE_OFFSET0	0x5280	1

MMIO Name	MMIO Offset	SIZE in DWords
SO_WRITE_OFFSET1	0x5284	1
SO_WRITE_OFFSET2	0x5288	1
SO_WRITE_OFFSET3	0x528C	1
PERF_CNT_1_DW0	0x91b8	1
PERF_CNT_1_DW1	0x91bc	1
PERF_CNT_2_DW0	0x91c0	1
PERF_CNT_2_DW1	0x91c4	1

Watchdog Timer Registers

These registers together implement a watchdog timer. Writing ones to the control register enables the counter, and writing zeros disables the counter. The second register is programmed with a threshold value which, when reached, signals an interrupt that then resets the counter to 0. Program the threshold value before enabling the counter or extremely frequent interrupts may result.

Note: The counter itself is not observable. It increments with the main render clock.

Programming Notes: When watch dog timer is enabled, HW does not trigger any kind of idle sequences. SW must enable and disable watch dog timer for any given workload within the same command buffer dispatch. SW must disable watch dog timer around semaphore waits and wait for events commands so that HW can trigger appropriate idle sequence for power savings.

PR_CTR - Render Watchdog Counter

PR_CTR_THRSH - Render Watchdog Counter Threshold

Command Stream Front End

Interrupt Control Registers

The Interrupt Control Registers described in this section all share the same bit definition. The bit definition is as follows:

Bit Definition for Interrupt Control Registers - Render

Bit Definition for Interrupt Control Registers - Blitter

Bit Definition for Interrupt Control Registers Media#1 (VDBOX1)

Bit Definition for Interrupt Control Registers Media#2 (VDBOX2)

Bit Definition for Interrupt Control Registers- Video Enhancement(VDBOX)

The following table specifies the settings of interrupt bits stored upon a "Hardware Status Write" due to ISR changes:

Bit	Interrupt Bit	ISR Bit Reporting Via Hardware Status Write (When Unmasked Via HWSTAM)
9	Reserved	
8	**Context Switch Interrupt.** Set when a context switch has just occurred.	Not supported to be unmasked.
7	**Page Fault.** This bit is set whenever there is a pending PPGTT (page or directory) fault. This interrupt is for handling Legacy Page Fault interface for all Command Streamers (BCS, RCS, VCS, VECS). When Fault Repair Mode is enabled, Interrupt mask register value is not looked at to generate interrupt due to page fault. Please refer to vol1c "Page Fault Support" section for more details.	Set when event occurs, cleared when event cleared. Not supported to be unmasked.
6	**Media Decode Pipeline Counter Exceeded Notify Interrupt.** The counter threshold for the execution of the media pipeline is exceeded. Driver needs to attempt hang recovery.	Not supported to be unmasked. Only for Media Pipe.
5	**L3 Parity interrupt**	Only for Render Pipe
4	**Flush Notify Enable**	0
3	**Master Error**	Set when error occurs, cleared when error cleared.
2	Reserved	
0	**User Interrupt**	0

RCS_HWSTAM - Render Hardware Status Mask Register

RCS_IMR - Render Interrupt Mask Register

VCS_HWSTAM - VCS Hardware Status Mask Register

VCS_IMR - VCS Interrupt Mask Register

BCS_HWSTAM - BCS Hardware Status Mask Register

BCS_IMR - BCS Interrupt Mask Register

VECS_HWSTAM - VECS Hardware Status Mask Register

VECS_IMR - VECS Interrupt Mask Register

Hardware-Detected Error Bit Definitions (for EIR EMR ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that is common to the EIR, EMR, and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR will cause the Master Error bit in the ISR to be set. EIR bits will remain set until the appropriate bit(s) in the EIR is cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following structures describe the Hardware-Detected Error bits:

RCS Hardware-Detected Error Bit Definitions Structure

BCS Hardware-Detected Error Bit Definitions Structure

VCS Hardware-Detected Error Bit Definitions Structure

VECS Hardware-Detected Error Bit Definitions Structure

The following are the EIR, EMR, and ESR registers:

RCS_EIR - Error Identity Register

RCS_EMR - Error Mask Register

RCS_ESR - Error Status Register

BCS_EIR - BCS Error Identity Register

BCS_EMR - BCS Error Mask Register

BCS_ESR - BCS Error Status Register

VCS_EIR - VCS Error Identity Register

VCS_EMR - VCS Error Mask Register

VCS_ESR - VCS Error Status Register

VECS_EIR - VECS Error Identity Register

VECS_EMR - VECS Error Mask Register

VECS_ESR - VECS Error Status Register

Blitter Engine Command Streamer (BCS)

The BCS (Blitter Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the Blitter Engine. It is responsible for fetching, decoding, and dispatching of data packets (Blitter Commands) to the front end interface module of Blitter Engine.

Logic Functions Included
• MMIO register programming interface. • DMA action for fetching of ring data from memory. • Management of the Head pointer for the Ring Buffer. • Decode of ring data and sending it to the blit engine. • Handling of user interrupts. • Flushing the Blitter Engine. • Handle NOP.
• DMA action for fetching of execlists from memory. • Handling of ring context switch interrupt.

The register programming bus is a DWord interface bus that is driven by the configuration master. The BCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x22000 to 0x224FF. The Blitter, Render and Media Engines use semaphore to synchronize their operations.

BCS operates completely independent of the othe render and media command streams.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside BCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (8 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards Blit Engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Watchdog Timer Registers

These are the Watchdog Timer registers:

BCS_CNTR - BCS Counter for the Blitter Engine

BCS_CTR_THRSH - BCS Watchdog Counter Threshold

Hardware-Detected Error Bit Definitions (for EIR EMR ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that are common to the EIR, EMR, and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR causes the Master Error bit in the ISR to be set. EIR bits remain set until the appropriate bits in the EIR are cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following structure describes the Hardware-Detected Error bits:

BCS Hardware-Detected Error Bit Definitions Structure

The following are the EIR, EMR, and ESR registers:

BCS_EIR - BCS Error Identity Register

BCS_EMR - BCS Error Mask Register

BCS_ESR - BCS Error Status Register

Logical Context Support

The following are the Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register

CTXT_SR_CTL - Context Save/Restore Control Register

BCS_SYNC_FLIP_STATUS - BCS Wait for event and Display flip flags Register

SYNC_FLIP_STATUS - Wait For Event and Display Flip Flags Register

SYNC_FLIP_STATUS_1 - Wait For Event and Display Flip Flags Register 1

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

BB_ADDR_DIFF - Batch Address Difference Register

BB_OFFSET - Batch Offset Register

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register

MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1

MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

INDIRECT_CTX - Indirect Context Pointer

INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Mode Registers

The following are Mode Registers:

BCS_CXT_SIZE - BCS Context Sizes

BCS_MI_MODE - BCS Mode Register for Software Interface

BLT_MODE - Blitter Mode Register

BCS_INSTPM - BCS Instruction Parser Mode Register

Programming Note	
Context:	Mode Registers in Blitter Engine Command Streamer

- The BCS_INSTPM register is used to control the operation of the BCS Instruction Parser. Certain classes of instructions can be disabled (ignored) – often useful for detecting performance bottlenecks. Also, "Synchronizing Flush" operations can be initiated – useful for ensuring the completion (vs. only parsing) of rendering instructions.
- All Reserved bits are implemented.

BCS_EXCC - BCS Execute Condition Code Register

BCS_IDLEDLY - BCS Idle Switch Delay

BCS_SEMA_WAIT_POLL - BCS Semaphore Polling Interval on Wait

BCS_RESET_CTRL - BCS Reset Control Register

BCS_PREEMPTION_HINT - BCS_PREEMPTION_HINT

BCS_PREEMPTION_HINT_UDW - BCS_PREEMPTION_HINT_UDW

BCS_CTXID_PREEMPTION_HINT - BCS Context ID Preemption Hint

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

MI Commands for Blitter Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the blitter graphics processing engine. The term "for Blitter Engine" in the title has been added to differentiate this chapter from a similar one describing the MI commands for the Media Decode Engine and the Rendering Engine.

The commands detailed in this chapter are used across products. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for product specific summary.

MI_NOOP

MI_ARB_CHECK

MI_ARB_ON_OFF

MI_BATCH_BUFFER_START

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as "Command Parser Master Error" interrupt to SW. Privilege access violation checks in HW can be disabled by setting "Privilege Check Disable" bit in BLT_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
MI_UPDATE_GTT	Command is converted to NOOP.
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_STORE_DATA_INDEX	Command is converted to NOOP.
MI_STORE_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Register write to a **Privileged Register** is discarded.
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** is discarded.

MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_SEMAPHORE_SIGNAL	Command is converted to NOOP.
MI_ARB_ON_OFF	Command is converted to NOOP.
MI_DISPLAY_FLIP	Command is converted to NOOP.
MI_FLUSH_DW	Command is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.

Parsing one of the commands in the table above from a non-privileged batch buffer flags an error and converts the command to a NOOP.

The following table lists the non-privileged registers that can be written to from a non-secure batch buffer executed from Render Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	22600h	32
BCS_SWCTRL	22200h	32

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_DISPLAY_FLIP

MI_LOAD_SCAN_LINES_EXCL

MI_LOAD_SCAN_LINES_INCL

MI_FLUSH_DW

MI_MATH

MI_REPORT_HEAD

MI_STORE_DATA_IMM

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_ATOMIC

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_REG

MI_LOAD_REGISTER_MEM

MI_STORE_REGISTER_MEM

MI_SUSPEND_FLUSH

MI_UPDATE_GTT

MI_USER_INTERRUPT

MI_WAIT_FOR_EVENT

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_WAIT

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring buffer. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing a privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as "Command Parser Master Error" interrupt to SW. Privilege access violation checks in HW can be disabled by setting "Privilege Check Disable" bit in BLT_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
MI_UPDATE_GTT	Command is converted to NOOP.
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_STORE_DATA_INDEX	Command is converted to NOOP.
MI_STORE_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Register write to a **Privileged Register** is discarded.
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** is discarded.
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_SEMAPHORE_SIGNAL	Command is converted to NOOP.
MI_ARB_ON_OFF	Command is converted to NOOP.
MI_DISPLAY_FLIP	Command is converted to NOOP.
MI_FLUSH_DW	Still send flush down, Post-Sync Operation is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.

Parsing one of the commands in the table above from a non-privileged batch buffer will flag an error and convert the command to a NOOP.

The following table lists the non-privileged registers that can be written to from a non-privileged batch buffer executed from Blitter Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
BCS_GPR	22600h	32
BCS_SWCTRL	22200h	32

Video Command Streamer (VCS)

The VCS (Video Command Streamer) unit primarily serves as the software programming interface between the O/S driver and the MFD Engine. It is responsible for fetching, decoding, and dispatching of data packets (Media Commands with the header DWord removed) to the front end interface module of MFX Engine.

Its logic functions include:

- MMIO register programming interface

  DMA action for fetching of execlists and ring data from memory
- Management of the Head pointer for the Ring Buffer
- Decode of ring data and sending it to the appropriate destination: AVC, VC1, or MPEG2 engine
- Handling of user interrupts

  Handling of ring context switch interrupt
- Flushing the MFX Engine
- Handle NOP

The register programming (RM) bus is a DWord interface bus that is driven by the Gx Command Streamer. The VCS unit only claims memory mapped I/O cycles that are targeted to its range of 0x4000 to 0x4FFFF. The Gx and MFX Engines use semaphore to synchronize their operations.

VCS operates completely independent of the Gx CS.

The simple sequence of events is as follows: a ring (say PRB0) is programmed by a memory-mapped register write cycle. The DMA inside VCS is kicked off. The DMA fetches commands from memory based on the starting address and head pointer. The DMA requests cache lines from memory (one cacheline CL at a time). There is guaranteed space in the DMA FIFO (16 CL deep) for data coming back from memory. The DMA control logic has copies of the head pointer and the tail pointer. The DMA increments the head pointer after making requests for ring commands. Once the DMA copy of the head pointer becomes equal to the tail pointer, the DMA stops requesting.

The parser starts executing once the DMA FIFO has valid commands. All the commands have a header DWord packet. Based on the encoding in the header packet, the command may be targeted towards AVC/VC1/MPEG2 engine or the command parser. After execution of every command, the actual head pointer is updated. The ring is considered empty when the head pointer becomes equal to the tail pointer.

Watchdog Timer Registers

The following registers are defined as Watchdog Timer registers:

VCS_CNTR - VCS Counter for the bit stream decode engine

VCS_THRSH - VCS Threshold for the counter of bit stream decode engine

VCS Hardware - Detected Error Bit Definitions (for EIR EMR ESR)

This section defines the Hardware-Detected Error bit definitions and ordering that are common to the EIR, EMR, and ESR registers. The EMR selects which error conditions (bits) in the ESR are reported in the EIR. Any bit set in the EIR causes the Master Error bit in the ISR to be set. EIR bits remain set until the appropriate bits in the EIR are cleared by writing the appropriate EIR bits with 1 (except for the unrecoverable bits described below).

The following links describe the Hardware-Detected Error bits.

VCS Hardware-Detected Error Bit Definitions

VCS_EIR - VCS Error Identity Register

VCS_EMR - VCS Error Mask Register

VCS_ESR - VCS Error Status Register

Logical Context Support

This section contains the registers for Logical Context Support.

BB_STATE - Batch Buffer State Register

CTXT_SR_CTL - Context Save/Restore Control Register

BB_ADDR_DIFF - Batch Address Difference Register

BB_ADDR - Batch Buffer Head Pointer Register

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register

BB_OFFSET - Batch Offset Register

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register

MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1

MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

INDIRECT_CTX - Indirect Context Pointer

INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Mode Registers

The following are Mode Registers:

VCS_CXT_SIZE - VCS Context Sizes]

VCS_MI_MODE - VCS Mode Register for Software Interface

MFX_MODE - Video Mode Register

VCS_INSTPM - VCS Instruction Parser Mode Register

VCS_IDLEDLY - VCS Idle Switch Delay

VCS_RESET_CTRL - VCS Reset Control Register

VCS_CTXID_PREEMPTION_HINT - VCS Context ID Preemption Hint

VCS_PREEMPTION_HINT - VCS_PREEMPTION_HINT

VCS_PREEMPTION_HINT_UDW

VCS_SEMA_WAIT_POLL - VCS Semaphore Polling Interval on Wait

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

Registers in Media Engine

This topic describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. The functions performed by some of these registers are discussed in more detail in the Memory Interface Functions, Memory Interface Instructions, and Programming Environment chapters.

The registers detailed in this chapter are used across multiple projects and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

GFX Pending TLB Cycles Information Registers

The following registers contain information about cycles that did not complete their TLB translation.

Information is organized as 64 entries, where each entry has a valid and ready bit, collapsed into separate registers.

VCS_TIMESTAMP - VCS Reported Timestamp Count

VCS_CTX_TIMESTAMP - VCS Context Timestamp Count

Memory Interface Commands for Video Codec Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

MI_ARB_CHECK

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_BATCH_BUFFER_START

MI_FLUSH_DW

MI_LOAD_REGISTER_IMM

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_REG

MI_MATH

MI_NOOP

MI_REPORT_HEAD

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_WAIT

MI_STORE_REGISTER_MEM

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_SUSPEND_FLUSH

MI_USER_INTERRUPT

MI_UPDATE_GTT

MI_LOAD_REGISTER_MEM[CHV]

MI_ATOMIC

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. Command Privilege Violation Error is logged in Error identity register of command streamer which gets propagated as "Command Parser Master Error" interrupt to SW. Privilege access violation checks in HW can be disabled by setting "Privilege Check Disable" bit in MFX_MODE register. When privilege access checks are disabled HW will execute the Privilege command as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
MI_UPDATE_GTT	Command is converted to NOOP.
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_STORE_DATA_INDEX	Command is converted to NOOP.
MI_STORE_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled Register write to a **Privileged Register** will be discarded.
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** will be discarded.
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.
MI_SEMAPHORE_SIGNAL	Command is converted to NOOP.
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_ARB_ON_OFF	Command is converted to NOOP.
MI_FLUSH_DW	Still send flush down, Post-Sync Operation is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_SEMAPHORE_SIGNAL	Command is converted to NOOP.

Parsing one of the commands in the table above from a non-privileged batch buffer flags an error and converts the command to a NOOP.

The following table lists the non-privileged registers that can be written to from a non-secure batch buffer

executed from Video Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
VCS_GPR	12600h	32
VCS_GPR (2nd VCS)	1C600h	32
MFC_VDBOX1	12800h	64
MFC_VDBOX2	1C800h	64

Video Enhancement Engine Command Interface

The following topics describe the Video Enhancement Engine Command Interface.

VECS_RINGBUF — Ring Buffer Registers

The following are Ring Buffer Registers:

RING_BUFFER_TAIL - Ring Buffer Tail

RING_BUFFER_HEAD - Ring Buffer Head

RING_BUFFER_START - Ring Buffer Start

RING_BUFFER_CTL - Ring Buffer Control

UHPTR - Pending Head Pointer Register

Watchdog Timer Registers

The following are Watchdog Timer Registers:

VECS_CNTR - VECS Counter for the Video Enhancement Engine

VECS_CTR_THRSH - VECS Threshold for the Counter of Video Enhancement Engine

Logical Context Support

The following are Logical Context Support Registers:

BB_ADDR - Batch Buffer Head Pointer Register

BB_ADDR_UDW - Batch Buffer Upper Head Pointer Register

SBB_ADDR - Second Level Batch Buffer Head Pointer Register

SBB_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Register

BB_STATE - Batch Buffer State Register

CTXT_SR_CTL - Context Save/Restore Control Register

VECS_TIMESTAMP - VECS Reported Timestamp CountCHV

VECS_CTX_TIMESTAMP - VECS Context Timestamp Count

BB_ADDR_DIFF - Batch Address Difference Register

BB_OFFSET - Batch Offset Register

RING_BUFFER_HEAD_PREEMPT_REG - RING_BUFFER_HEAD_PREEMPT_REG

BB_PREEMPT_ADDR - Batch Buffer Head Pointer Preemption Register

BB_PREEMPT_ADDR_UDW - Batch Buffer Upper Head Pointer Preemption Register

SBB_PREEMPT_ADDR - Second Level Batch Buffer Head Pointer Preemption Register

SBB_PREEMPT_ADDR_UDW - Second Level Batch Buffer Upper Head Pointer Preemption Register

MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1

MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

DISPLAY_MESSAGE_FORWARD_STATUS - Display Message Forward Status Register

FORCE_TO_NONPRIV - FORCE_TO_NONPRIV

INDIRECT_CTX - Indirect Context Pointer

INDIRECT_CTX_OFFSET - Indirect Context Offset Pointer

BB_PER_CTX_PTR - Batch Buffer Per Context Pointer

Mode Registers

The following are Mode Registers:

VECS_MI_MODE — VECS Mode Register for Software Interface

VEBOX_MODE - Video Mode Register

VECS_INSTPM—VECS Instruction Parser Mode Register

HWS_PGA - Hardware Status Page Address Register

Hardware Status Page Layout

VECS_RESET_CTRL - VECS Reset Control Register

VECS_PREEMPTION_HINT

VECS_PREEMPTION_HINT_UDW

VECS_CTXID_PREEMPTION_HINT - VECS Context ID Preemption Hint

VECS_SEMA_WAIT_POLL - VECS Semaphore Polling Interval on Wait

MI Commands for Video Enhancement Engine

This chapter describes the formats of the "Memory Interface" commands, including brief descriptions of their use. The functions performed by these commands are discussed fully in the *Memory Interface Functions* Device Programming Environment chapter.

This chapter describes MI Commands for the Video Codec Engine.

The commands detailed in this chapter are used across product families. However, slight changes may be present in some commands (i.e., for features added or removed), or some commands may be removed entirely. Refer to the *Preface* chapter for details.

MI_ARB_CHECK

MI_BATCH_BUFFER_END

MI_CONDITIONAL_BATCH_BUFFER_END

MI_BATCH_BUFFER_START

MI_FLUSH_DW

MI_LOAD_REGISTER_IMM

MI_LOAD_REGISTER_REG

MI_MATH

MI_NOOP

MI_REPORT_HEAD

MI_SEMAPHORE_SIGNAL

MI_SEMAPHORE_WAIT

MI_ATOMIC

MI_STORE_REGISTER_MEM

MI_STORE_DATA_IMM

MI_STORE_DATA_INDEX

MI_SUSPEND_FLUSH

MI_USER_INTERRUPT

MI_UPDATE_GTT

MI_COPY_MEM_MEM

MI_LOAD_REGISTER_MEM

User Mode Privileged Commands

A subset of the commands are privileged. These commands may be issued only from a privileged batch buffer or directly from a ring. Batch buffers in GGTT memory space are privileged and batch buffers in PPGTT memory space are non-privileged. On parsing a privileged command from a non-privileged batch buffer, a Command Privilege Violation Error is flagged and the command is dropped. A Command Privilege Violation Error is logged in the Error identity register of the command streamer which gets propagated as "Command Parser Master Error" interrupt to SW. Privilege access violation checks in HW can be disabled by setting "Privilege Check Disable" bit in VEBOX_MODE register. When privilege access checks are disabled HW executes privileged commands as expected.

User Mode Privileged Commands

User Mode Privileged Command	Function in Non-Privileged Batch Buffers
MI_UPDATE_GTT	Command is converted to NOOP.
MI_STORE_DATA_IMM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_STORE_DATA_INDEX	Command is converted to NOOP.
MI_STORE_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_BATCH_BUFFER_START	Command when executed from a batch buffer can set its "Privileged" level to its parent batch buffer or lower. Chained or Second level batch buffer can be "Privileged" only if the parent or the initial batch buffer is "Privileged". This is HW enforced.
MI_LOAD_REGISTER_IMM	Command is converted to NOOP if the register accessed is privileged.
MI_LOAD_REGISTER_MEM	Command is converted to NOOP if **Use Global GTT** is enabled. Register write to a **Privileged Register** will be discarded.
MI_LOAD_REGISTER_REG	Register write to a **Privileged Register** will be discarded.
MI_ATOMIC	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_COPY_MEM_MEM	Command is converted to NOOP if **Use Global GTT** is used for source or destination address.
MI_SEMAPHORE_SIGNAL	Command is converted to NOOP.
MI_SEMAPHORE_WAIT	Command is converted to NOOP if **Use Global GTT** is enabled.
MI_ARB_ON_OFF	Command is converted to NOOP.
MI_FLUSH_DW	Still send flush down, Post-Sync Operation is converted to NOOP if **Use Global GTT** or Use "Store Data Index" is enabled.
MI_CONDITIONAL_BATCH_BUFFER_END	Command is converted to NOOP if **Use Global GTT** is enabled.

Parsing one of the commands in the table above from a non-privileged batch buffer will flag an error and convert the command to a NOOP.

The table below lists the non-privileged registers that can be written to from a non-secure batch buffer executed from the Render Command Streamer.

User Mode Non-Privileged Registers

MMIO Name	MMIO Offset	Size in DWords
VECS_GPR	1A600h	32

Preemption

Preemption is a means by which HW is instructed to stop executing an ongoing workload and switch to the new workload submitted. Preemption flows are different based on the mode of scheduling.

Ring Buffer Scheduling

In Ring Buffer mode of scheduling SW triggers preemption by programming UHPTR (Updated Head Pointer Register) register with a valid head pointer. UHPTR contains head pointer and head pointer valid bit; the head pointer is valid only when the head pointer valid bit is set.

HW triggers preemption on a preemptable command on detecting Head Pointer Valid bit asserted in the UHPTR register. Following preemption HW updates its current head pointer with the Head Pointer from the UHPTR and starts execution; i.e all the commands from current head pointer to the updated head pointer are skipped by HW. HW samples the head pointer and the batch buffer address on preemption and updates them to the RING_BUFFER_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR respectively. RING_BUFFER_HEAD_PREEMPT_REG and BB_PREEMPT_ADDR provide the graphics memory address of the preemptable command on which last preemption has occurred. HW resets the head pointer valid bit in UHPTR upon completion of preemption.

Programming Notes:

Preemption is not supported for Media Workloads. Hence preemption can be achieved only on Command Buffer boundaries. Media Command Buffers must be bracketed with MI_ARB_OFF and MI_ARB_ON commands to avoid preemption of media command buffers.

Example:

```
Ring Buffer
.
.
.
MI_ARB_ON_OFF     // OFF
MI_BATCH_START    // Media Workload
MI_ARB_ON_OFF     // ON
MI_ARB_CHK        // Preemptable command outside media command buffer.
.
.
End Ring Buffer
```

The following tables list the Preemptable Commands in the Ring Buffer mode of scheduling:

Engine (below)	Preemptable Commands				
	MI_ARB_CHECK	3DPRIMITIVE	GPGPU_WALKER	PIPE_CONTROL ***	MEDIA STATE FLUSH
Render	AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	N/A	N/A	N/A	N/A
Media	AP	N/A	N/A	N/A	N/A
VideoEnhancement	AP	N/A	N/A	N/A	N/A

AP: Allow preemption on UHPTR valid and arbitration enabled. Arbitration can be enabled/disabled using MI_ARB_ON_OFF command.

ExecList Scheduling

In ExecList mode of scheduling SW triggers preemption by submitting a new pending execlist to ELSP (ExecList Submit Port). HW triggers preemption on a preemptable command on detecting the availability of the new pending execlist, following preemption context switch happens to the newly submitted execlist. As part of the context switch preempted context state is saved to the preempted context LRCA, context state contains the details such that on resubmission of the preempted context HW can resume execution from the point where it was preempted.

Example:

```
Ring Buffer

MI_ARB_ON_OFF   // OFF
MI_BATCH_START  // Media Workload
MI_ARB_ON_OFF   // ON
MI_ARB_CHK      // Preemptable command outside media command buffer.
```

The following tables list the Preemptable Commands in ExecList mode of scheduling:

Engine (below)	Preemptable Commands							
	MI_ARB_CHECK	Element Boundary	Semaphore Wait	Wait for Event	3DPRIMITIVE	GPGPU_WALKER	PIPE_CONTROL ***	MEDIA STATE FLUSH
Render	AP	AP	Unsuccessful & AP	Unsuccessful & AP	Object Level (if enabled *)	Mid-Thread (if enabled **)	PIPESEL-GPGPU MODE	PIPESEL-GPGPU MODE
Blitter	AP	AP	Unsuccessful & AP	Unsuccessful & AP	N/A	N/A	N/A	N/A
Media	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A
Video Enhancement	AP	AP	Unsuccessful & AP	N/A	N/A	N/A	N/A	N/A

Preemption is not supported for Media Workloads. Hence preemption can be achieved only on Command Buffer boundaries. Media Command Buffers must be bracketed with MI_ARB_OFF and MI_ARB_ON command to avoid preemption of media command buffers.

Table Notes:

AP - Allow Preemption if arbitration is enabled.

* 0x229c bit 11 determines whether the level of preemption is command or object level.

** 0x20E4 bits 2:1 determine the level of preemption for GPGPU workloads.

*** MI_ATOMIC and MI_SEMAPHORE_SIGNAL commands with Post Sync Op bit set are treated as PIPE_CONTROL command with Post Sync Operation as Atomics or Semaphore Signal.

**** Any Header with the value [31:29] = "011", [28:27] = "00" OR "11" and [26:24] = "001". Refer to Graphics Command Formats.

Command Streamer (CS) ALU Programming

The command streamer implements a rudimentary Arithmetic Logic Unit (ALU) which supports basic arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64-bit operands.

The ALU has two 64-bit registers at the input, SRCA and SRCB, to which source operands are loaded. The ALU result is written to a 64-bit accumulator. The Zero Flag and Carry Flag are assigned based on the accumulator output.

See the ALU Programming section in the Render Engine Command Streamer, for a description of the ALU programming model. That model is the same for all command streamers that support ALU programming, but each command streamer uses different address offsets for the registers used. The following subsections describe the ALU registers in the Blitter command streamer.

CS ALU Programming and Design

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 7: 3D Media GPGPU

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Registers in Render Engine 1
Predication 2
Predicate Render Registers 2
MI_SET_PREDICATE 3
MI_PREDICATE 3
Predicated Rendering Support in HW 4
Registers in Render Engine 5
Context Save Registers 5
VF Instance Count Registers 5
Mode and Misc Ctrl Registers 6
Pipelines Statistics Counter Registers 7
AUTO_DRAW Registers 9
MMIO Registers for GPGPU Indirect Dispatch 9
CS ALU Programming and Design 10
CS ALU Programming and Design 10
Generic Purpose Registers 10
ALU BLOCK Diagram 11
Instruction Set 12
Instruction Format 12
LOAD Operation 12
Arithmetic/Logical Operations 13
STORE Operation 13
Summary for ALU 13
Summary of Instructions Supported 14
Table for ALU OPCODE Encodings 14
Table for Register Encodings 16
CS_GPR - Command Streamer General Purpose Registers 16
Memory Interface Commands for Rendering Engine 16
State Commands 17
STATE_BASE_ADDRESS 17
Synchronization of the 3D Pipeline 18

Top-of-Pipe Synchronization 18
End-of-Pipe Synchronization 18
Synchronization Actions 19
Writing a Value to Memory 19
PS_DEPTH_COUNT 20
Generating an Interrupt 20
Invalidating of Caches 20
PIPE_CONTROL Command 20
PIPE_CONTROL 21
Programming Restrictions for PIPE_CONTROL 22
Post-Sync Operation 23
Flush Types 24
Stall 24
Render Logical Context Data 25
Overall Context Layout 25
Context Layout 25
Register State Context 26
Register State Context 27
Ring Buffer 38
Ring Context 38
The Per-Process Hardware Status Page 39
Render Engine Power Context 40
Command Ordering Rules 42
PIPELINE_SELECT 43
PIPE_CONTROL 43
Common Pipeline State-Setting Commands 43
3D Pipeline-Specific State-Setting Commands 44
Media Pipeline-Specific State-Setting Commands 44
3DPRIMITIVE 44
MEDIA_OBJECT 44
Resource Streamer 45
Resource Streamer Sync Commands 45
Introduction 46

Glossary 46
Common Abbreviations 46
Theory of Operation 47
Resource Streamer Functions 47
Detailed Resource Streamer Operations 49
Introduction 49
Resource Streamer Operation Descriptions 49
Batch Processing 49
Context Save 49
HW Binding Table Image 50
Gather Push Constants Image 50
Push Constant Image 51
HW Binding Table Generation 52
Gather Push Constants 53
Constant Buffer Generation (not DX9) 54
Commands Actions in the RS 55
Resource Streamer Programming Guidelines 61
RS Interactions with the 3D Command Streamer 61
RS Interactions with Memory Requests 61
Fundamental Programming and Operational Assumptions 61
Non-Operational Activities 62
Hardware Binding Tables 62
3DSTATE_BINDING_TABLE_POOL_ALLOC [CHV, BSW] 63
Gather Constants 64
Dx9 Constant Buffer Generation 65
Vertex Shader Constant 66
Pixel Shader Constant 67
Shared Functions 68
3D Sampler 68
Sampling Engine 69
Texture Coordinate Processing 71
Texture Coordinate Normalization 71
Texture Coordinate Computation 71

Texel Address Generation 73
Level of Detail Computation (Mipmapping) 73
Base Level Of Detail (LOD) 74
LOD Bias 74
LOD Pre-Clamping 75
Min/Mag Determination 75
LOD Computation Pseudocode 76
Intra-Level Filtering Setup 77
MAPFILTER_NEAREST 78
MAPFILTER_LINEAR 78
Bilinear Filter Sampling 79
MAPFILTER_ANISOTROPIC 79
MAPFILTER_MONO 80
Inter-Level Filtering Setup 82
Texture Address Control 82
TEXCOORDMODE_MIRROR Mode 84
TEXCOORDMODE_MIRROR_ONCE Mode 85
TEXCOORDMODE_WRAP Mode 86
TEXCOORDMODE_CLAMP Mode 87
TEXCOORDMODE_CLAMPBORDER Mode 88
TEXCOORDMODE_CUBE Mode 88
Texel Fetch 89
Texel Chroma Keying 89
Chroma Key Testing 89
Chroma Key Effects 90
Shadow Prefilter Compare 90
Texel Filtering 91
Texel Color Gamma Linearization 91
Multisampled Surface Behavior 92
Multisample Control Surface 92
State 93
Surface State Fetch 93
Sampler State Fetch 93

State Caching 94
SURFACE_STATE 94
Sampler Output Channel Mapping 96
SURFACE_STATE for Deinterlace sample_8x8 and VME 102
SAMPLER_STATE 103
Border Color Programming for Integer Surface Formats 104
Messages 106
Message Descriptor and Execution Mask 106
Execution Mask 106
Message Descriptor 106
Message Header 107
Parameter Types 111
SIMD Payloads 112
Writeback Message 114
SIMD16 114
Return Format = 32-bit 114
Return Format = 16-bit 115
SIMD8/SIMD8D 116
Return Format = 32-bit 116
Return Format = 16-bit 117
SIMD4x2 118
Shared Functions – Data Port 119
Data Cache 120
Sampler Cache 121
Surfaces 121
Surface State Model 121
Stateless Model 121
Shared Local Memory (SLM) 122
Write Commit 123
ReadWrite Ordering 123
Accessing Buffers 124
Accessing Media Surfaces 125
Boundary Behavior 125

State 126
BINDING_TABLE_STATE 126
SURFACE_STATE 126
COLOR_PROCESSING_STATE 126
Messages 127
Global Definitions 127
Data Port Messages 127
Message Descriptor 130
Message Descriptor 130
Message Header 133
Write Commit Writeback Message 134
OWord Block ReadWrite 134
Message Descriptor 136
Message Payload (Write) 136
Writeback Message (Read) 137
Unaligned OWord Block Read 137
Message Descriptor 138
Writeback Message (Read) 138
OWord Dual Block ReadWrite 139
Message Descriptor 140
Message Payload 141
Additional Message Payload (Write) 141
Writeback Message (Read) 142
Media Block ReadWrite 142
Message Descriptor 144
Message Header 146
Message Payload (Write) 149
Writeback Message (Read) 150
DWord Scattered ReadWrite 150
Message Descriptor 151
Message Payload 151
Additional Message Payload (Write) 152
Writeback Message (Read) 152

Message Descriptor 153
Message Payload 154
Source Payload 155
Writeback Message 155
Byte Scattered ReadWrite 156
Message Descriptor 157
Message Payload 158
Additional Message Payload (Write) 158
Writeback Message (Read) 159
TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation 160
Typed Surface ReadWrite Message Descriptor 166
Typed Surface Read/Write Message Descriptor 167
Untyped Surface ReadWrite Message Descriptor 168
Typed Atomic Operation Message Descriptor 169
Typed Atomic Operation SIMD4x2 Message Descriptor 170
Untyped Atomic Operation Message Descriptor 171
Untyped Atomic Operation SIMD4x2 Message Descriptor 172
Atomic Counter Operation Message Descriptor 173
Atomic Counter Operation SIMD4x2 Message Descriptor 174
Message Header 175
Message Payload 175
SIMD16 Address Payload 176
SIMD16 Source Payload (Atomic Operation Message Only) 177
SIMD16 Source Payload (AOP_CMPWR8B Only) 178
SIMD16 Write Data Payload (Write Message Only) 179
SIMD8 Address Payload 180
SIMD8 Source Payload (Atomic Operation Message Only) 181
SIMD8 Write Data Payload (Write Message Only) 182
SIMD8 Write Data Payload (Tile W Write Message Only) 183
SIMD4x2 Address Payload 184
SIMD4x2 Source Payload (Atomic Operation Message Only) 184
SIMD4x2 Source Payload (AOP_CMPWR8B Only) 185
SIMD4x2 Write Data Payload (Write Message Only) 185

Writeback Message 185
SIMD8 DWORD Read 186
SIMD8 QWORD Read 186
SIMD16 Read 187
SIMD8 Read 188
SIMD8 Read (Tile W) 189
SIMD4x2 Read 190
SIMD16 Atomic Operation 191
SIMD16 Atomic Operation (AOP_CMPWR8B Only) 192
SIMD8 Atomic Operation 193
SIMD8 Atomic Operation (AOP_CMPWR8B Only) 194
SIMD4x2 Atomic Operation 195
SIMD4x2 Atomic Operation (AOP_CMPWR8B Only) 195
A64 Scattered Read or Write 196
Message Descriptor 197
Message Address Payload 198
Additional Message Payload (QWord Write) 199
Additional Message Payload (DWord Write) 201
Additional Message Payload (8 Byte Write) 202
Additional Message Payload (1 or 2 or 4 Byte Write) 203
Writeback Message (QWord Read) 203
Writeback Message (DWord Read) 204
Writeback Message (8 Byte Read) 206
Writeback Message (1 or 2 or 4 Byte Read) 206
A64 Untyped Atomic Operation 207
A64 Untyped Atomic Operation Message Descriptor 209
Message Payload 211
SIMD8 Address Payload 212
SIMD4x2 Address Payload 213
SIMD8 Source Payload (QWORD Atomic Operation Message Only) 214
SIMD8 Source Payload (DWORD Atomic Operation Message Only) 215
SIMD8 Source Payload (AOP_CMPWR16B Only) 216
SIMD8 Source Payload (AOP_CMPWR8B Only) 216

SIMD4x2 Source Payload (QWORD Atomic Operation Message Only) 217
SIMD4x2 Source Payload (DWORD Atomic Operation Message Only) 218
SIMD4x2 Source Payload (AOP_CMPWR16B Only) 218
SIMD4x2 Source Payload (AOP_CMPWR8B Only) 219
Writeback Message 219
SIMD8 Atomic Operation (QWord) 219
SIMD8 Atomic Operation (DWord) 220
SIMD8 Atomic Operation (AOP_CMPWR16B Only) 221
SIMD8 Atomic Operation (AOP_CMPWR8B Only) 222
SIMD4x2 Atomic Operation (QWord) 223
SIMD4x2 Atomic Operation (DWord) 223
SIMD4x2 Atomic Operation (AOP_CMPWR16B Only) 224
SIMD4x2 Atomic Operation (AOP_CMPWR8B Only) 224
A64 Untyped Atomic Float Add Operation 225
Untyped Atomic Float Add Operation Message Descriptor 225
Message Payload 226
SIMD8 Address Payload 226
SIMD8 Source Payload (QWORD Atomic Operation Message Only) 227
SIMD8 Source Payload (DWORD Atomic Operation Message Only) 228
Writeback Message 228
A64 Untyped Surface Read or Write Operation 228
Untyped Surface Read or Write Message Descriptor 230
Message Payload 230
SIMD8 Address Payload 231
SIMD4x2 Address Payload 232
SIMD8 DWORD Write Data Payload (Write Message Only) 233
SIMD4x2 DWORD Write Data Payload (Write Message Only) 234
Writeback Message 234
SIMD8 DWORD Read 235
SIMD4x2 DWORD Read 236
A64 Block Read or Write Operation 236
Message Descriptor 237
Message Header 238

Message Payload (OWord Write) 239
Writeback Message (OWord Read) 239
Writeback Message (Unaligned OWord Read) 239
Message Payload (Dual OWord Write) 240
Writeback Message (Dual Oword Read) 240
Message Payload (HWord Write) 241
Writeback Message (HWord Read) 241
Untyped Atomic Float Add Operation 241
Untyped Atomic Float Add Operation Message Descriptor 243
Message Header 244
Message Payload 245
SIMD16 Address Payload 245
SIMD8 Address Payload 246
SIMD16/SIMD8 DWORD Source Payload 246
SIMD16/SIMD8 QWORD Source Payload 247
Writeback Message 248
SIMD16 SIMD8 DWORD Atomic Float Add 248
SIMD16/SIMD8 QWORD Atomic Float Add 248
Read Surface Info 249
Message Descriptor 249
Address Payload 249
Writeback Message 249
Transpose Read 251
Message Descriptor 252
Message Header 252
Writeback Message 254
Scratch Block Read or Write 254
Message Descriptor 255
Message Header 256
Message Payload (Write) 256
Message Payload (Read) 256
Writeback Message (Read) 256
Memory Fence 257

Message Descriptor 258
Message Header 259
Writeback Message 259
Pixel Data Port 260
Cache Agents 260
Accessing Render Targets 260
Message Sequencing Summary 261
Single Source 262
Dual Source 263
Replicate Data 263
Multiple Render Targets (MRT) 263
Render Target Read and Write 263
SubspanPixel to Slot Mapping 267
Message Descriptor 270
Message Header 270
Message Header 271
Writeback Message (Read) 272
Header for SIMD8_IMAGE_WRITE 273
Source 0 Alpha Payload 277
oMask Payload 278
Color Payload SIMD16 Single Source 279
Color Payload 279
Color Payload SIMD8 Single Source 281
Color Payload SIMD16 Replicated Data 283
Color Payload SIMD8 Dual Source 284
Render Target Read and Write 286
Message Header 289
Shared Functions Pixel Interpolater 290
Messages 290
Initiating Message 290
Message Descriptor 290
“Per Message Offset” Message Descriptor 292
“Sample Position Offset” Message Descriptor 292

"Centroid Position" and "Per Slot Offset" Message Descriptor 292
Message Payload for Most Messages 293
SIMD8 Per Slot Offset Message Payload 293
SIMD16 Per Slot Offset Message Payload 294
Writeback Message 295
SIMD8 295
SIMD16 296
Shared Functions - Unified Return Buffer (URB) 298
URB Size 298
URB Access 298
State 299
FF_SYNC Messages 299
FF_SYNC Message Header 299
FF_SYNC Writeback Message 300
URB Messages 301
Execution Mask 302
Message Descriptor 302
URB_WRITE and URB_READ 304
Message Header 304
URB_WRITE_HWORD Write Data Payload 306
URB_NOSWIZZLE 306
URB_INTERLEAVED 307
URB_READ_HWORD Writeback Message 308
URB_NOSWIZZLE 308
URB_INTERLEAVED 309
URB_WRITE_OWORD Write Data Payload 310
URB_NOSWIZZLE 310
URB_INTERLEAVED 311
URB_READ_OWORD Writeback Message 312
URB_NOSWIZZLE 312
URB_INTERLEAVED 313
URB_ATOMIC 314
Message Header 314

Writeback Message 315
URB_SIMD8_Write and URB_SIMD8_Read 316
Message Descriptor 316
Message Header 317
Per Slot Offset Message Phase 318
Channel Mask Message Phase 319
Write Data Payload 320
Writeback Message 321
Message Gateway 322
Messages 323
Message Descriptor 323
OpenGateway Message 324
Message Payload 325
Writeback Message to Requester Thread 326
CloseGateway Message 326
Message Payload 327
Writeback Message to Requester Thread 328
ForwardMsg Message 328
Message Payload 329
Writeback Message to Requester Thread 331
Writeback Message to Recipient Thread 331
GetTimeStamp Message 331
Message Payload 332
Writeback Message to Requester Thread 333
BarrierMsg Message 334
Message Payload 335
Writeback Message to Requester Thread 336
Broadcast Writeback Message 336
MMIOReadWrite Message 336
Message Payload 337
Writeback Message to Requester Thread (MMIO Read Only) 337
Media Sampler 338
Shared Functions – Video Motion Estimation 338

Theory of Operation 338
Shape Decision 338
Minor Shape Decision Prior to FME 339
Major Shape Decision Prior to FME 342
Shape Update after FME 342
Final Code Decision after BME 342
Early Decisions 343
Surfaces 344
State 345
BINDING_TABLE_STATE 345
SURFACE_STATE 345
VME_STATE 345
VME_SEARCH_PATH_LUT_STATE 345
Software Interface – Bspec Highlights 350
Message Structure Overview 350
IME and IDM Message Descriptor 351
Input GRFs 352
Input Message Phases by Type 353
Output GRFs 353
Output Message Phases by Type 354
Binding Table Pointers 355
Progressive Content 355
Interlaced Content 358
RDE Packet Mapping 360
Glossary of Messages 361
Universal Input Message Phases 361
SIC Input Message Phases 384
IME Input Message Phases 389
FBR Input Message Phases 394
IDM Input Message Phases 396
Return Data Message Phases 397
IME StreamOut 408
IDM Stream-Out 408

IDM16x16 Streamout Message Format 409
IDM8x8 Streamout Message Format 410
Sample_8x8 State 412
SURFACE_STATE for Deinterlace, sample_8x8, and VME 412
SAMPLER_STATE for Sample_8x8 Message 412
Media Object Dispatch Pseudocode 413
Calculate Residual Blocks Pseudocode 413
Dispatch Media Object Pseudocode 414
SIMD32/64 Payload 414
SIMD32 Payload 414
SIMD64 Payload 415
Vertical Block Number Restrictions 421
Payload Parameter Definition 422
SIMD32_64 Message Descriptor 424
SIMD32_64 Message Header 424
Message Header 425
SIMD32_64 Payload Parameter Definition 427
SIMD32_64 Message Types 427
Writeback Message 427
SIMD32 428
Sample_unorm* 428
Cache_flush 432
Sample_8x8 Writeback Messages 433
“16 bit Full” Output Format Control Mode 434
Sampler_8x8 – Writeback Message for Convolve and 1 Pixel Convolution and 1D Vertical Convolve 439
Sampler_8x8 – Writeback Message for MinMaxFilter 440
"16 Bit Full" Output Format Control Mode 440
“8 Bit Full” Output Format Control Mode: 442
Sampler_8x8 – Writeback Message for MinMax 443
Sampler_8x8 – Writeback Message for Dilate or Erode 444
Sampler_8x8 – Writeback Message for Centroid 444
Sampler_8x8 – Writeback Message for BoolCentroid/BoolSum 445
Sampler_8x8 – Writeback Message for AVS 445

"16 bit Full" Output Format Control Mode 445
"16 Bit Chrominance Downsampled" Output Format Control Mode 447
"8 Bit Full" Output Format Control Mode 449
"8 Bit Chrominance Downsampled" Output Format Control Mode 450
SIMD32 Surface State 452
SIMD32 Sampler State 452
3D Pipeline Stages 452
3D Pipeline-Level State 453
Statistics 455
Statistics Gathering 455
3D Pipeline Geometry 457
Block Diagram 457
3D Primitives Overview 458
Vertex Data Overview 466
Vertex Positions 466
Clip Space Position 467
NDC Space Position 467
Screen-Space Position 467
Vertex Fetch (VF) Stage 468
State 468
Control State 468
Index Buffer (IB) State 468
Vertex Buffers (VB) State 469
VERTEXDATA Buffers – SEQUENTIAL Access 469
VERTEXDATA Buffers – RANDOM Access 470
INSTANCEDATA Buffers 471
Vertex Definition State 472
Input Vertex Definition 472
3D Primitive Command 473
Functions 473
Input Assembly 473
Vertex Assembly 473
Vertex Cache 474

Input Data: Push Model vs. Pull Model 475
Generated IDs 475
3D Primitive Processing 476
Index Buffer Access 476
Vertex Element Data Path 477
FormatConversion 479
DestinationFormatSelection 482
Dangling Vertex Removal 482
Vertices Generated 483
Objects Generated 483
Vertex Shader (VS) Stage 484
State 484
Functions 484
Vertex Shader Cache (VS$) 484
SIMD4x2 VS Thread Request Generation 486
SIMD4x2 VS Thread Execution 486
VS Thread Dispatch Masks 487
Vertex Output 487
Thread Termination 487
Primitive Output 487
Statistics Gathering 488
Payloads 488
SIMD4x2 Payload 488
SIMD8 Payload 490
Hull Shader (HS) Stage 493
State 493
Functions 494
Patch Object Staging 494
HS Thread Execution 494
HS Thread Dispatch Mask 494
Patch URB Entry (Patch Record) Output 494
Please Provide Title 494
DOMAIN_POINT Structure 497

Statistics Gathering 498
HS Invocations 498
Payloads 498
SINGLE_PATCH Payload 498
Tessellation Engine (TE) Stage 503
State 503
Functions 503
Patch Culling 503
Tessellation Factor Limits 504
Partitioning 504
Domain Types and Output Topologies 504
QUAD Domain Tessellation 505
TRI Domain Tessellation 507
ISOLINE Domain Tessellation 508
Domain Shader (DS) Stage 509
State 509
Functions 509
SIMD4x2 Thread Execution 509
Statistics Gathering 510
Payloads 510
SIMD4x2 Payload 510
SIMD8 Payload 513
Geometry Shader (GS) Stage 516
GS Stage Overview 516
State 516
Functions 516
Object Staging 516
Thread Request Generation 517
Object Vertex Ordering 517
Thread Execution 522
Thread Execution 522
GS URB Entry 522
GS URB Entry - Output Vertex Count 523

GS Output Topologies 525
GS Output StreamID 525
Primitive Output 526
Statistics Gathering 526
Payloads 526
Thread Payload High-Level Layout 526
SIMD 4x2 Thread Payload 527
SIMD8 Thread Payload 534
Thread Request Generation 541
Thread Control Information 542
Thread Payload Generation 543
Fixed Payload Header 543
Extended Payload Header 546
Payload URB Data 546
Stream Output Logic (SOL) Stage 548
State 548
Functions 550
Input Buffering 550
Stream Output Function 553
Stream Output Buffers 554
Rendering Disable 554
Statistics 555
3D Pipeline Rasterization 555
Common Rasterization State 555
3D Pipeline – CLIP Stage Overview 556
Clip Stage – 3D Clipping 556
Fixed Function Clipper 556
Concepts 557
CLIP Stage Input 557
State 557
VUE Readback 558
VertexClipTest Function 559
Object Staging 562

Partial Object Removal 562
ClipDetermination Function 562
ClipMode State 564
NORMAL ClipMode 565
CLIP_ALL ClipMode 565
CLIP_NON_REJECT ClipMode 565
REJECT_ALL ClipMode 565
ACCEPT_ALL ClipMode 565
Object Pass-Through 566
Primitive Output 567
Other Functionality 567
Statistics Gathering 568
CL_INVOCATION_COUNT 568
3D Pipeline - Strips and Fans (SF) Stage 569
Inputs from CLIP 569
Attribute Setup/Interpolation Process 570
Attribute Setup/Interpolation Process 570
Outputs to WM 570
Primitive Assembly 571
Point List Decomposition 575
Line List Decomposition 576
Line Strip Decomposition 576
Triangle List Decomposition 578
Triangle Strip Decomposition 579
Triangle Fan Decomposition 580
Polygon Decomposition 581
Rectangle List Decomposition 581
Object Setup 583
Invalid Position Culling (Pre/Post-Transform) 583
Viewport Transformation 583
Destination Origin Bias 583
Point Rasterization Rule Adjustment 584
Drawing Rectangle Offset Application 585

Point Width Application 587
Rectangle Completion 587
Vertex XY Clamping and Quantization 588
Degenerate Object Culling 589
Triangle Orientation (Face) Culling 589
Scissor Rectangle Clipping 590
Viewport Extents Test 591
Line Rasterization 591
Zero-Width (Cosmetic) Line Rasterization 591
GIQ (Diamond) Sampling Rules – Legacy Mode 592
GIQ (Diamond) Sampling Rules – DX10 Mode 593
Non-Antialiased Wide Line Rasterization 595
Anti-Aliased Line Rasterization 596
SF Pipeline State Summary 597
3DSTATE_RASTER [CHV, BSW] 597
3DSTATE_SF 598
Attribute Interpolation Setup 603
Attribute Swizzling 603
Interpolation Modes 604
Point Sprites 604
Barycentric Attribute Interpolation 606
Depth Offset 606
Other SF Functions 606
Statistics Gathering 606
Windower (WM) Stage 607
Overview 607
Inputs from SF to WM 608
Windower Pipelined State 608
3DSTATE_WM 608
3DSTATE_SAMPLE_MASK 613
Rasterization 619
Drawing Rectangle Clipping 620
Line Rasterization 620

Coverage Values for Anti-Aliased Lines 620
3DSTATE_AA_LINE_PARAMS [CHV, BSW] 620
Line Stipple 621
Polygon (Triangle and Rectangle) Rasterization 621
Polygon Stipple 622
Multisampling 623
Multisample ModesState 623
Other WM Functions 629
Statistics Gathering 629
Pixel 629
Early Depth/Stencil Processing 630
Depth Offset 630
Early Depth Test/Stencil Test/Write 631
Software-Provided PS Kernel Info 632
Hierarchical Depth Buffer 632
Depth Buffer Clear 634
Depth Buffer Resolve 635
Hierarchical Depth Buffer Resolve 636
Optimized Depth Buffer Clear and/or Stencil Buffer Clear 637
Optimized Depth Buffer Resolve 638
Optimized Hierarchical Depth Buffer Resolve 638
Separate Stencil Buffer 638
DepthStencil Buffer State 639
Pixel Shader Thread Generation 639
3DSTATE_PS 640
Pixel Grouping (Dispatch Size) Control 642
Multisampling Effects on Pixel Shader Dispatch 644
MSDISPMODE_PERPIXEL Thread Dispatch 644
MSDISPMODE_PERSAMPLE Thread Dispatch 644
PS Thread Payload for Normal Dispatch 647
PS Thread Payload for Normal Dispatch 647
Pixel Backend 661
Color Calculator (Output Merger) 661

Overview 661
Alpha Coverage 662
Alpha Test 662
Depth Coordinate Offset 663
Stencil Test 664
Depth Test 665
Pre-Blend Color Clamping 665
Pre-Blend Color Clamping When Blending is Disabled 666
Pre-Blend Color Clamping When Blending is Enabled 666
Color Buffer Blending 667
Post-Blend Color Clamping 669
Dithering 669
Logic Ops 670
Buffer Update 671
Stencil Buffer Updates 671
Depth Buffer Updates 672
Color Gamma Correction 672
Color Buffer Updates 672
Pixel Pipeline State Summary 672
COLOR_CALC_STATE 672
3DSTATE_BLEND_STATE_POINTERS 673
3DSTATE_DEPTH_STENCIL_STATE_POINTERS 673
COLOR_CALC_STATE 673
DEPTH_STENCIL_STATE 673
BLEND_STATE 673
CC_VIEWPORT 673
Other Pixel Pipeline Functions 674
Statistics Gathering 674
MCS Buffer for Render Target(s) 674
Render Target Fast Clear 677
Render Target Resolve 677
Media GPGPU Pipeline 679
Media GPGPU Pipeline 679

Programming the GPGPU Pipeline 679
GPGPU Thread Limits 680
GPGPU Commands 680
GPGPU Command Workarounds 680
GPGPU Indirect Thread Dispatch 681
GPGPU Context Switch 682
GPGPU Context Switch 683
GPGPU Context Switch Workarounds 686
Media GPGPU Payload Limitations 686
Synchronization of the Media/GPGPU Pipeline 687
Mode of Operations 688
GPGPU Thread R0 Header 688
GPGPU Mode 690
Automatic Thread Generation 690
Thread Payload 691
Execution Masks 692
URB Management 693
Indirect Payload Storage 694
MEDIA_OBJECT_GRPID 697
Starting Offset for a Thread Group ID 697
Thread Group Tracking 698
Shared Local Memory Allocation 699
Software Managed Shared Local Memory 699
Automatic Barrier Management 699
Dispatch Payload 700
Generic Media 700
Product Evolution 702
Media and General Purpose Pipeline 705
Introduction 705
Terminologies 705
Hardware Feature Map in Products 706
Media Pipeline Overview 708
Generic Mode 709

Programming Media Pipeline 710
Command Sequence 710
Command Sequence 710
Parameterized Media Walker 712
Walker Parameter Description 713
Basic Parameters for the Local Loop 714
MbAff-Like Special Case in Local Loop 716
Global Loop 717
Walker Algorithm Description 718
Barriers and Shared Local Memory 722
Flexible Dispatch of Local Loop 723
Scoreboard Control 724
AVC-Style Dependency Example 725
Interface Descriptor Selection 727
VC1-Style Dependency Example 729
Multiple Slice Considerations 729
Interrupt Latency 729
Thread Spawner Unit 730
Root Threads and Child Threads 731
Root Threads 731
URB Handles 732
Root to Child Responsibilities 732
Multiple Simultaneous Roots 732
Synchronized Root Threads 733
Deadlock Prevention 733
Child Thread Life Cycle 734
Arbitration between Root and Child Threads 735
Persistent Root Thread 735
Media State Model 736
Media State and Primitive Commands 737
Media State and Primitive Command Workarounds 737
Media Messages 738
Thread Payload Messages 738

Generic Mode Root Thread 739
Root Thread from MEDIA_OBJECT_PRT 740
Root Thread from MEDIA_OBJECT_WALKER 742
MEDIA_OBJECT_GRPID and MEDIA_OBJECT_WALKER with Groups Payload 742
Thread Spawn Message 744
Message Descriptor 745
Message Payload 745
L3 Cache and URB 747
L3 Cache and URB 747
Vol 1i L3$/URB 748
Overview (8th Generation) 748
L3$ Cache Configuration 749
Blocks(s) Overview 749
Size of L3 for 8th Generation 750
ECC Protection 750
L3$ Theory of Operation 750
Atomics 752
L3 Coherency 757
Thread level Coherency 757
Thread Group Coherency 757
GPU/IA level Coherency 757
Coherency Usage Models 758
Fixed Func. Producing (URB) 758
Fixed Func. Producing (Push Constants) 758
EUs Producing via HDC 758
Invalidation and Flushes 759
Node Architecture Impact 759
Command Streamer Flows 759
Non-IA Coherent Flows 760
Top of the Pipe Invalidations 760
End of the Pipe Flush - Pipeline Flush 760
End of the Pipe Flush - GT3/GT4 760
End of the Pipe Flush - GT2/GT1 760

IA-Coherent Flows 760
EU/Thread Flows 760
Global Invalidation 761
Power Management Invalidation 761
L3 Allocation and Programming 762
Non-SLM Mode Allocation 762
SLM Mode Allocation 763
Shared Local Memory (SLM) 764
Save and Restore Requirements 765
SLM Context Save in SuperQ 766
6SLM Context Restore in SuperQ 766
SLM Context Save in L3 767
SLM Context Restore in L3 767
EU Overview 768
CoIssue/Dual Issue: 769
Thread scheduling: 769
Primary Usage Models 770
AOS and SOA Data Structures 771
SIMD4 Mode of Operation 773
SIMD4x2 Mode of Operation 774
SIMD16 Mode of Operation 776
SIMD8 Mode of Operation 778
Messages 779
Message Payload Containing a Header 780
Writebacks 780
Message Delivery Ordering Rules 781
Execution Mask and Messages 781
End-Of-Thread (EOT) Message 782
Performance 783
Message Description Syntax 783
Message Errors 784
Registers and Register Regions 786
Register Files 786

GRF Registers 787
ARF Registers 788
ARF Registers Overview 788
Access Granularity 789
Null Register 790
Address Register 791
Accumulator Registers 795
Flag Register 800
Channel Enable Register 801
Message Control Registers 802
Example 803
SP Register 803
State Register 805
Control Register 811
Notification Registers 817
IP Register 819
TDR Registers 820
Performance Registers 823
Flow Control Registers 825
Immediate 829
Region Parameters 830
Region Addressing Modes 834
Direct Register Addressing 834
Register-Indirect Register Addressing with a 1x1 Index Region 835
Register-Indirect Register Addressing with a Vx1 Index Region 836
Register-Indirect Register Addressing with a VxH Index Region 837
Access Modes 839
Execution Data Type 840
Register Region Restrictions 841
Destination Operand Description 850
Destination Region Parameters 850
SIMD Execution Control 851
Predication 851

No Predication 853
Predication with Horizontal Combination 853
Predication with Vertical Combination 855
End of Thread 855
Assigning Conditional Flags 856
Destination Hazard 859
Non-present Operands 860
Instruction Prefetch 860
ISA Introduction 861
Introducing the Execution Unit 862
EU Terms and Acronyms 865
Execution Units (EUs) 870
EU Changes by Processor Generation 871
EU Notation 874
Execution Environment 875
EU Data Types 875
Fundamental Data Types 875
Numeric Data Types 876
Integer Numeric Data Types 876
Floating-Point Numeric Data Types 878
Packed Signed Half-Byte Integer Vector 880
Packed UnSigned Half-Byte Integer Vector 882
Packed Restricted Float Vector 883
Floating Point Modes 885
IEEE Floating Point Mode 885
Partial Listing of Honored IEEE-754 Rules 885
Complete Listing of Deviations or Additional Requirements vs IEEE-754 886
Min Max of Floating Point Numbers 887
Alternative Floating Point Mode 889
Floating-Point Support 891
Floating-Point Types and Values 891
Not a Number (NaN) Formats 892
Floating-Point Rounding Modes 893

Floating-Point Operations and Precision 894
Single Precision Floating-Point Rounding to Integral Values 894
Floating-Point to Integer Conversion 894
Integer to Floating-Point Conversion 896
Floating-Point Min/Max Operations 896
IEEE Floating-Point Exceptions 897
Signaling Floating-Point Exceptions 897
Invalid Operation Exception 898
Division by Zero Exception 899
Overflow Exception 899
Underflow Exception 900
Inexact Exception 901
Floating-Point Compare Operations 901
Type Conversion 906
Float to Integer 906
Integer to Integer with Same or Higher Precision 906
Integer to Integer with Lower Precision 907
Integer to Float 907
Double Precision Float to Single Precision Float [CHV, BSW] 908
Single Precision Float to Double Precision Float [CHV, BSW] 908
Exceptions 909
Exception-Related Architecture Registers 910
System Routine 911
Invoking the System Routine 911
Returning to the Application Thread 912
System IP (SIP) 913
System Routine Register Space 913
System Scratch Memory Space 914
Conditional Instructions Within the System Routine 914
Use of NoDDClr 915
Exception Descriptions 916
Illegal Opcode 916
Undefined Opcodes 916

Software Exception 916
Context Save and Restore 916
Events That Do Not Generate Exceptions 918
Illegal Instruction Format 918
Malformed Message 918
GRF Register Out of Bounds 918
Hung Thread 918
Instruction Fetch Out of Bounds 918
FPU Math Errors 919
Computational Overflow 919
System Routine Example 920
Instruction Set Summary 923
Instruction Set Characteristics 923
SIMD Instructions and SIMD Width 923
Instruction Operands and Register Regions 923
Instruction Execution 924
Instruction Formats 924
Native Instruction Layouts 929
Instruction Fields 933
EU Compact Instructions 951
EU Compact Instruction Format 951
EU Instruction Compaction Tables 957
Opcode Encoding 963
Move and Logic Instructions 963
Flow Control Instructions 964
Miscellaneous Instructions 965
Parallel Arithmetic Instructions 965
Vector Arithmetic Instructions 966
Special Instructions 968
Native Instruction BNF 969
Instruction Groups 969
Destination Register 970
Source Register 971

Address Registers 972
Register Files and Register Numbers 972
Relative Location and Stack Control 973
Regions 974
Types 974
Write Mask 974
Swizzle Control 974
Immediate Values 974
Predication and Modifiers 975
Instruction Options 976
Instruction Set Summary Tables 977
Accumulator Restrictions 980
Instruction Set Reference 983
Conventions 983
Pseudo Code Format 983
General Macros and Definitions 983
Evaluate Write Enable 984
EUISA Instructions 985
Round Instructions 987
rndd – Round Down 988
rnde – Round to Nearest or Even 989
rndu – Round Up 990
rndz – Round to Zero 991
Send Message 991
Send Message [CHV, BSW] 992
EUISA Structures 995
EUISA Enumerations 996
EU Programming Guide 998
Assembler Pragmas 998
Declarations 998
Defaults and Defines 998
Example Pragma Usages 999
Assembly Programming Guideline 1001

Usage Examples....1002
Vector Immediate....1002
Supporting DirectX 10 Pixel Shader Indexing....1002
Supporting OpenGL Vertex Shader Instruction SWZ....1003
Destination Mask for DP4 and Destination Dependency Control....1004
Null Register as the Destination....1004
Use of LINE Instruction....1005
Mask for SEND Instruction....1006
Channel Enables for Extended Math Unit....1006
Channel Enables for Scratch Memory....1008
Flow Control Instructions....1010
Execution Masking....1010
Branching....1011
Fast-If....1011
Cascade Branching....1012
Compound Branches....1013
Looping....1014
Indexed Jump....1019

Registers in Render Engine

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. Refer to each registers description and related feature for more information on each individual bit. Unless noted otherwise within the description of the bit must only be updated following a flush to ensure the pipeline is idle.

The registers detailed in this chapter are used across the CHV, BSW family of products and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

Predication

Predicate Render Registers

MI_PREDICATE_SRC0 - Predicate Rendering Temporary Register0

MI_PREDICATE_SRC1 - Predicate Rendering Temporary Register1

MI_PREDICATE_DATA - Predicate Rendering Data Storage

MI_PREDICATE_RESULT - Predicate Rendering Data Result

MI_PREDICATE_RESULT_1 - Predicate Rendering Data Result 1

MI_PREDICATE_RESULT_2 - Predicate Rendering Data Result 2

MI_SET_PREDICATE

MI_SET_PREDICATE is a command that allows the driver to conditionally choose to execute or skip a command during execution time.

Below is a table of command(s) that are allowed to be programmed when the MI_SET_PREDICATE PREDICATE ENABLE field allows any type of predication. If a command is not in the table, then it will have undefined behavior when executed with predication enabled:

MI_SET_PREDICATE

Command
3DSTATE_URB_VS
3DSTATE_URB_HS
3DSTATE_URB_DS
3DSTATE_URB_GS
3DSTATE_PUSH_CONSTANT_ALLOC_VS
3DSTATE_PUSH_CONSTANT_ALLOC_HS
3DSTATE_PUSH_CONSTANT_ALLOC_DS
3DSTATE_PUSH_CONSTANT_ALLOC_GS
3DSTATE_PUSH_CONSTANT_ALLOC_PS
MI_LOAD_REGISTER_IMM
3DSTATE_WM_HZ_OP
MEDIA_VFE_STATE
MEDIA_OBJECT
MEDIA_OBJECT_WALKER
MEDIA_INTERFACE_DESCRIPTOR_LOAD

MI_PREDICATE

The MI_PREDICATE command is used to control the Predicate state bit, which in turn can be used to enable/disable the processing of 3DPRIMITIVE commands.

MI_PREDICATE

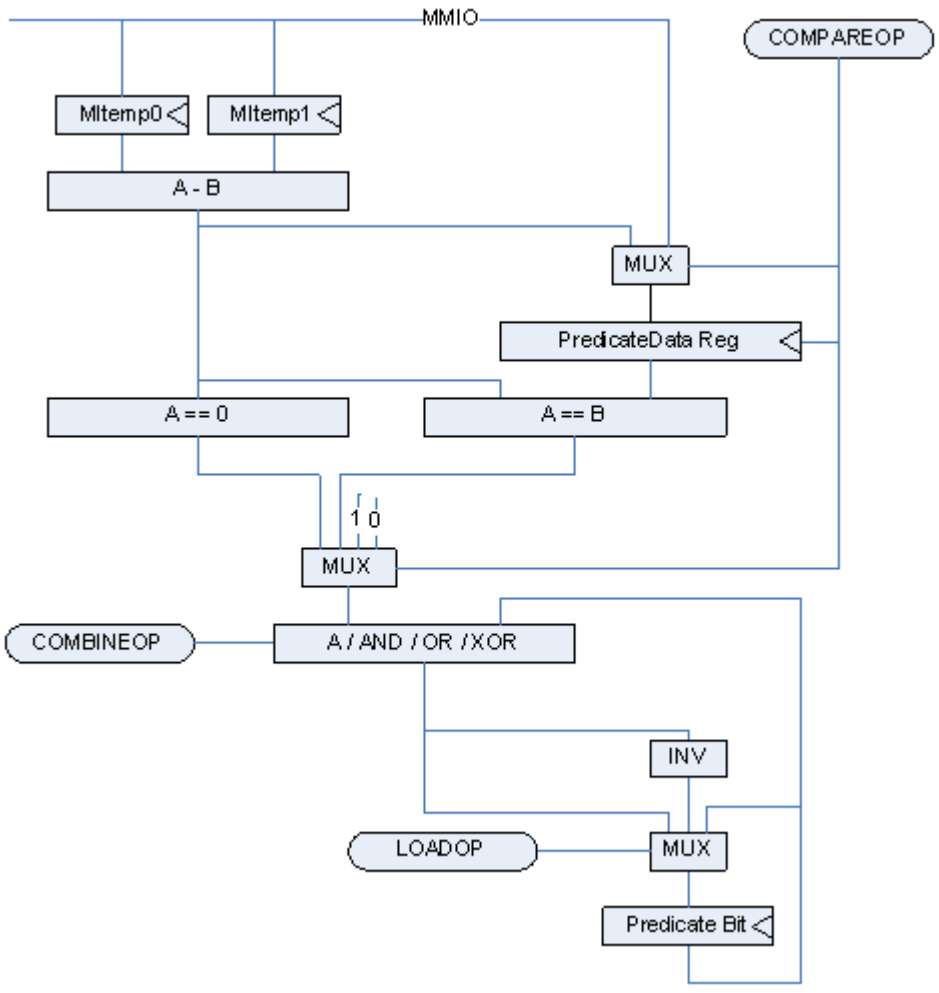
Predicated Rendering Support in HW

DX10 defines predicated rendering, where sequences of rendering commands can be discarded based on the result of a previous predicate test. A new state bit, Predicate, has been added to the command stream. In addition, a PredicateEnable bit is added to 3DPRIMITIVE. When the PredicateEnable bit is set, the command is ignored if the Predicate state bit is set.

A new command, MI_PREDICATE, is added. It contains several control fields which specify how the Predicate bit is generated.

Refer to the diagram below and the command description (linked above) for details.

MI_PREDICATE Function



MI_LOAD_REGISTER_MEM commands can be used to load the MItemp0, MItemp1, and PredicateData registers prior to MI_PREDICATE. To ensure the memory sources of the MI_LOAD_REGISTER_MEM commands are coherent with previous 3D_PIPECONTROL store-DWord operations, software can use the new **Pipe Control Flush Enable** bit in the PIPE_CONTROL command.

Registers in Render Engine

This chapter describes the memory-mapped registers associated with the Memory Interface, including brief descriptions of their use. Refer to each registers description and related feature for more information on each individual bit. The registers detailed in this chapter are used across the CHV, BSW family of products and are extentions to previous projects. However, slight changes may be present in some registers (i.e., for features added or removed), or some registers may be removed entirely. These changes are clearly marked within this chapter.

Context Save Registers

VF Instance Count Registers

VF Instance Count Register Set		
Register Type:	MMIO_VF	
Address:	08300h - 08384h	
Default Value:	0000 0000h	
Access:	RO	
Size:	1088 bits	
Description:	Set of Registers for storing the index count values. In case of preempted drawcalls, these register store index count/number per element. For the non-preempted drawcalls, the values stored are ignored upon restore. These are saved as part of render context.	
DWord	**Bits**	**Description**
0	31:0	**Index Count 0.** Index Count value for Element 0. Format: U32
1	31:0	**Index Count 1.** Index Count value for Element 1. Format: U32
...	31:0	...
33	31:0	**Index Count 33.** Index Count value for Element 33. Format: U32

Mode and Misc Ctrl Registers

This section contains various registers for controls and modes.

MI_MODE - Render Mode Register for Software Interface

FF_MODE - Thread Mode Register

GFX_MODE - Graphics Mode Register

GT_MODE - GT Mode Register

SAMPLER_MODE - SAMPLER Mode Register

CACHE_MODE_0 - Cache Mode Register 0

CACHE_MODE_1 - Cache Mode Register 1

GAFS_MODE - Mode Register for GAFS

FBC_RT_BASE_ADDR_REGISTER - FBC_RT_BASE_ADDR_REGISTER

FBC_RT_BASE_ADDR_REGISTER_UPPER - FBC_RT_BASE_ADDR_REGISTER_UPPER

L3CNTLREG - L3 Control Register

B/D/F/Type:

Address Offset: 0x7034

Default Value: 60000060h

Access: RW; RO;

Size: 32 bit

Below Register provides GT2 based L3 sizes.

For GT1 – all sizes need to be multiplied by 0.5.

For GT3 – all sizes need to be multiplied by 2.

For GT4 – all sizes need to be multiplied by 3.

All L3 ways have to be included in the programming to ensure that no ways are left out.

L3CNTLREG - L3 Control Register

GLOBAL_CLEAR_VALUE_0 - Global Clear Value Register 0

GLOBAL_CLEAR_VALUE_1 - Global Clear Value Register 1

GLOBAL_CLEAR_VALUE_2 - Global Clear Value Register 2

GLOBAL_CLEAR_VALUE_3 - Global Clear Value Register 3

Pipelines Statistics Counter Registers

These registers keep continuous count of statistics regarding the 3D pipeline. They are saved and restored with context but should not be changed by software except to reset them to 0 at context creation time. Write access to the statistics counter in this section must be done through MI_LOAD_REGISTER_IMM, MI_LOAD_REGISTER_MEM, or MI_LOAD_REGISTER_REG commands in ring buffer or batch buffer. These registers may be read at any time; however, to obtain a meaningful result, a pipeline flush just prior to reading the registers is necessary to synchronize the counts with the primitive stream.

IA_VERTICES_COUNT - IA Vertices Count

IA_PRIMITIVES_COUNT - Primitives Generated By VF

VS_INVOCATION_COUNT - VS Invocation Counter

HS_INVOCATION_COUNT - HS Invocation Counter

DS_INVOCATION_COUNT - DS Invocation Counter

GS_INVOCATION_COUNT - GS Invocation Counter

GS_PRIMITIVES_COUNT - GS Primitives Counter

CL_INVOCATION_COUNT - Clipper Invocation Counter

PS_INVOCATION_COUNT - PS Invocation Count

PS_INVOCATION_COUNT_SLICE0 - PS Invocation Count for Slice0

PS_INVOCATION_COUNT_SLICE1 - PS Invocation Count for Slice1

PS_INVOCATION_COUNT_SLICE2 - PS Invocation Count for Slice2

PS_DEPTH_COUNT

PS_DEPTH_COUNT_SLICE0 - PS Depth Count for Slice0

PS_DEPTH_COUNT_SLICE1 - PS Depth Count for Slice1

PS_DEPTH_COUNT_SLICE2 - PS Depth Count for Slice2

PS_DEPTH_COUNT_SLICE3 - PS Depth Count for Slice3

TIMESTAMP - Reported Timestamp Count

SO_NUM_PRIMS_WRITTEN[0:3] - Stream Output Num Primitives Written Counter

SO_PRIM_STORAGE_NEEDED[0:3] - Stream Output Primitive Storage Needed Counters

SO_WRITE_OFFSET[0:3] - Stream Output Write Offsets

WMHWCLRVAL - Window Hardware Generated Clear Value

CS_CTX_TIMESTAMP- CS Context Timestamp Count:

This register provides a mechanism to obtain cumulative run time of a GPU context on HW.

CS_CTX_TIMESTAMP - CS Context Timestamp Count

Diagram below details on when CS_CTX_TIMESTAMP reun time, save/restored during a GPGPU context switch flow.

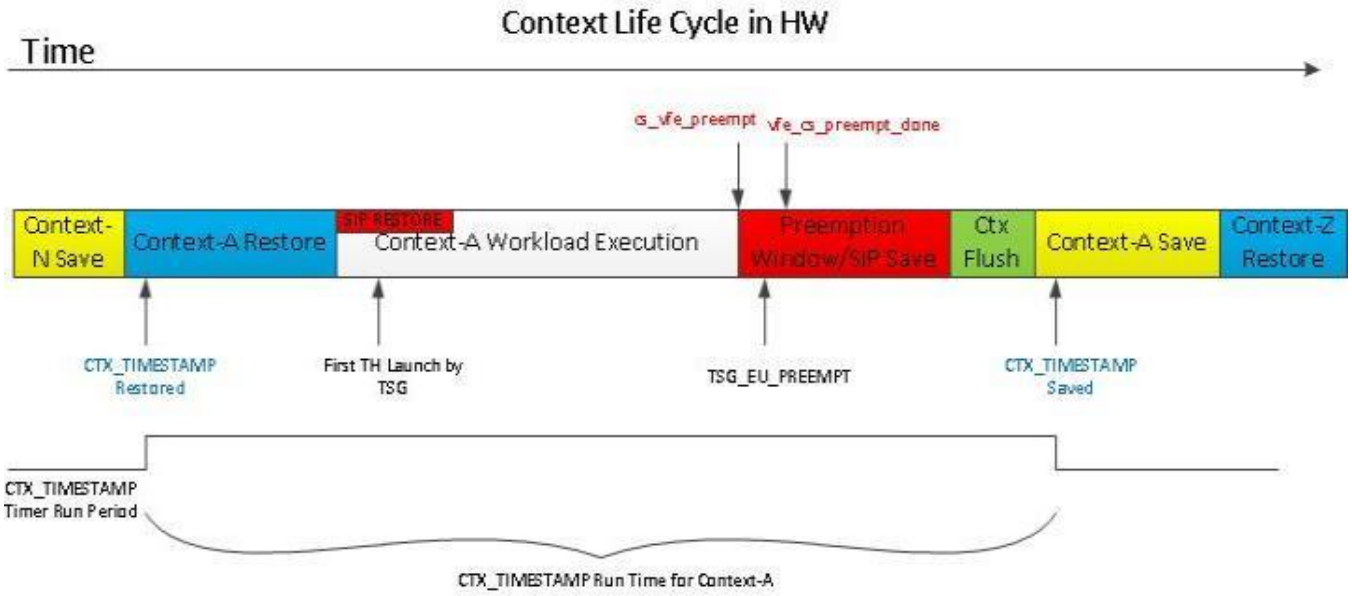


Fig: CTX_TIMESTAMP fucntionality during context execution

AUTO_DRAW Registers

3DPRIM_END_OFFSET - Auto Draw End Offset

3DPRIM_START_VERTEX - Load Indirect Start Vertex

3DPRIM_VERTEX_COUNT - Load Indirect Vertex Count

3DPRIM_INSTANCE_COUNT - Load Indirect Instance Count

3DPRIM_START_INSTANCE - Load Indirect Start Instance

3DPRIM_BASE_VERTEX - Load Indirect Base Vertex

MMIO Registers for GPGPU Indirect Dispatch

These registers are normally written with the MI_LOAD_REGISTER_MEMORY command rather than from the CPU.

GPGPU_DISPATCHDIMX - GPGPU Dispatch Dimension X

GPGPU_DISPATCHDIMY - GPGPU Dispatch Dimension Y

GPGPU_DISPATCHDIMZ - GPGPU Dispatch Dimension Z

TS_GPGPU_THREADS_DISPATCHED - Count Active Channels Dispatched

CS ALU Programming and Design

Command streamer implements a rudimentary ALU which supports basic Arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64bit operands. ALU has two 64bit registers at the input SRCA and SRCB to which the operands should be loaded on which operations will be performed and outputted to a 64 bit Accumulator. Zero Flag and Carry Flag are set based on accumulator output.

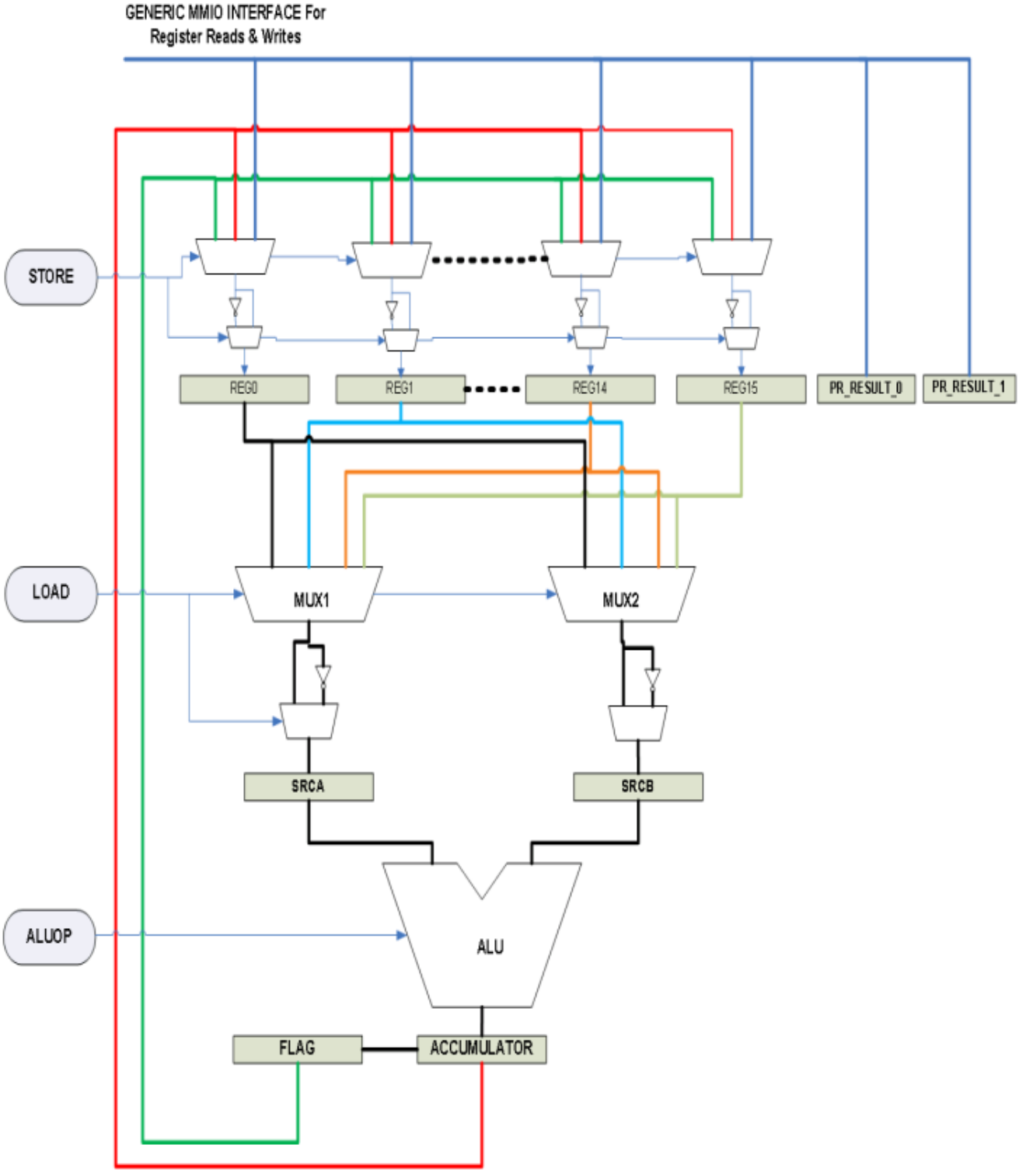
CS ALU Programming and Design

Command streamer implements a rudimentary ALU which supports basic Arithmetic (Addition and Subtraction) and logical operations (AND, OR, XOR) on two 64bit operands. ALU has two 64bit registers at the input SRCA and SRCB to which the operands should be loaded on which operations will be performed and outputted to a 64 bit Accumulator. Zero Flag and Carry Flag are set based on accumulator output.

Generic Purpose Registers

Command streamer implements sixteen 64 bit General Purpose Registers which are MMIO mapped. These registers can be accessed similar to any other MMIO mapped registers through LRI, SRM, LRR, LRM or CPU access path for reads and writes. These registers will be labeled as R0, R1, … R15 throughout the discussion. Refer table in the B-spec update section mapping these registers to corresponding MMIO offset. A selected GPR register can be moved to SRCA or SRCB register using “LOAD” instruction. Outputs of the ALU, Accumulator, ZF and CF can be moved to any of the GPR using “STORE” instruction.

ALU BLOCK Diagram



Instruction Set

The instructions supported by the ALU can be broadly categorized into three groups:

- To move data from GPR to SRCA/SRCB – LOAD instruction.
- To move data from ACCUMULATOR/CF/ZF to GPR – STORE Instruction.
- To do arithmetic/Logical operations on SRCA and SRCB of ALU - ADD/SUB/AND/XOR/OR.

Instruction Format

Each instruction is one Dword in size and consists of an ALU OPCODE, OPERAND1 and OPERAND2 in the format shown below.

ALU OPCODE	Operand-1	Operand-2
12 bits	10 bits	10 bits

LOAD Operation

The LOAD instruction moves the content of the destination register (Operand2) into the source register (Operand1). The destination register can be any of the GPR (R0, R1, ..., R15) and the source registers are SRCA and SRCB of the ALU. This is the only means SRCA and SRCB can be programmed.

LOAD has different flavors, wherein one can load the inverted version of the source register into the destination register or a hard coded value of all Zeros and All ones.

```
 // Loads any of Reg0 to Reg15 into the SRCA or SRCB registers of ALU.

     LOAD <SRCA, SRCB>, <REG0..REG15>

 // Loads inverted (bit wise) value of the mentioned Reg0 to 15 into SRCA or SRCB registers
of ALU.

     LOADINV <SRCA, SRCB>, <REG0..REG15>

 // Loads "0" into SRCA or SRCB

     LOAD0 <SRCA, SRCB>

 // Loads "1" into SRCA or SRCB

     LOAD1 <SRCA, SRCB>
```

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
LOAD	SRCA/SRCB	R0,R1..R15
LOADINV	SRCA/SRCB	R0,R1..R15
LOAD0	SRCA/SRCB	N/A
LOAD1	SRCA/SRCB	N/A

Arithmetic/Logical Operations

ADD, SUB, AND, OR, and XOR are the Arithmetic and Logical operations supported by Arithmetic Logic Unit (ALU). When opcode corresponding to a logical operation is performed on SRCA and SRCB, the result is sent to ACCUMULATOR (ACCU), CF and ZF. Note that ACCU is 64-bit register. A NOOP when submitted to the ALU doesn't do anything, it is meant for creating bubble or kill cycles.

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
ADD	N/A	N/A
SUB	N/A	N/A
AND	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A
NOOP	N/A	NA

STORE Operation

The STORE instruction moves the content of the destination register (Operand1) into the source register (Operand2). The destination register can be accumulator (ACCU), CF or ZF and GPR (R0, R1 ..R15). STORE has different flavors, wherein one can load the inverted version of the source register into destination register via STOREINV. When CF or ZF are stored, the same value is replicated on all 64 bits.

```
// Loads ACCMULATOR or Carry Flag or Zero Flag in to any of the generic registers
// Reg0 to Reg16. In case of CF and ZF same value is replicated on all the 64 bits.

    STORE   <R0.. R15>, <ACCU, CF, ZF>

// Loads inverted (ACCMULATOR or Carry Flag or Zero Flag) in to any of the
// generic registers Reg0 to Reg15.

    STOREINV <R0.. R15>, <ACCU, CF, ZF>
```

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
STORE	R0,R1..R15	ACCU/ZF/CF
STOREINV	R0, R1.. R15	ACCU/ZF/CF

Summary for ALU

Total Opcodes Supported: 12

Total Addressable Registers as source or destination: 21

- 16 GPR (R0, R1 …R15)
- 1 ACCU
- 1ZF
- 1CF
- SRCA, SRCB

Summary of Instructions Supported

31 20	19 10	9 0
Opcode	**Operand1**	**Operand2**
LOAD	SRCA/SRCB	REG0..REG15
LOADINV	SRCA/SRCB	REG0..REG15
LOAD0	SRCA/SRCB	N/A
LOAD1	SRCA/SRCB	N?A
ADD	N/A	N/A
SUB	N/A	N/A
AND	N/A	N/A
OR	N/A	N/A
XOR	N/A	N/A
NOOP	N/A	N/A
STORE	REG0..REG15	ACCU/CF/ZF
STOREINV	REG0..REG15	ACCU/CF/ZF

Table for ALU OPCODE Encodings

ALU OPCODE	OPCODE ENCODING
NOOP	0x000
LOAD	0x080
LOADINV	0x480
LOAD0	0x081
LOAD1	0x481
ADD	0x100
SUB	0x101
AND	0x102
OR	0x103
XOR	0x104
STORE	0x180
STOREINV	0x580

In the above mentioned table, ALU Opcode Encodings look like random numbers. The rationale behind those encodings is because the ALU Opcode is further broken down into sub-sections for ease-of-design implementation.

PREFIX		OPCODE		SUBOPCODE	
11	10	9	7	6	0
PREFIX VALUE		**Description**			
0		Regular			
1		Invert			
OPCODE VALUE		**Description**			
0		NOOP			
1		LOAD			
2		ALU			
3		STORE			

ALU OPCODE	ENCODING	PREFIX	OPCODE	SUBOPCODE
		10	9 7	6 0
NOOP	0x000	0	0	0
LOAD	0x080	0	1	0
LOADINV	0x480	1	1	0
LOAD0	0x081	0	1	1
LOAD1	0x481	1	1	1
ADD	0x100	0	2	0
SUB	0x101	0	2	1
AND	0x102	0	2	2
OR	0x103	0	2	3
XOR	0x104	0	2	4
STORE	0x180	0	3	0
STOREINV	0x580	1	3	0

Table for Register Encodings

Register	Register Encoding
R0	0x0
R1	0x1
R2	0x2
R3	0x3
R4	0x4
R5	0x5
R6	0x6
R7	0x7
R8	0x8
R9	0x9
R10	0xa
R11	0xb
R12	0xc
R13	0xd
R14	0xe
R15	0xf
SRCA	0x20
SRCB	0x21
ACCU	0x31
ZF	0x32
CF	0x33

CS_GPR - Command Streamer General Purpose Registers

Following are Command Streamer General Purpose Registers:

CS_GPR - CS General Purpose Register

Memory Interface Commands for Rendering Engine

MI_SET_CONTEXT

MI_TOPOLOGY_FILTER

MI_URB_CLEAR

MI_URB_ATOMIC_ALLOC

MI_LOAD_URB_MEM

MI_STORE_URB_MEM

State Commands

This section covers the following commands:

- **STATE_PREFETCH command.** The STATE_PREFETCH command is provided strictly as an optional mechanism to possibly enhance pipeline performance by prefetching data into the GPE's Instruction and State Cache (ISC).
- **STATE_SIP command**

STATE_PREFETCH

STATE_SIP

STATE_BASE_ADDRESS

The STATE_BASE_ADDRESS command sets the base pointers for subsequent state, instruction, and media indirect object accesses by the GPE. (See **Memory Access Indirection** for details.)

The following commands must be reissued following any change to the base addresses:

- 3DSTATE_PIPELINE_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- MEDIA_STATE_POINTERS

Execution of this command causes a full pipeline flush, thus its use should be minimized for higher performance.

STATE_BASE_ADDRESS

PIPELINE_SELECT

The **Pipeline Select** state is contained within the logical context.

Synchronization of the 3D Pipeline

Two types of synchronizations are supported for the 3D pipe: top of the pipe and end of the pipe. Top of the pipe synchronization really enforces the read-only cache invalidation. This synchronization guarantees that primitives rendered after such synchronization event fetches the latest read-only data from memory. End of the pipe synchronization enforces that the read and/or read-write buffers do not have outstanding hardware accesses. These are used to implement read and write fences as well as to write out certain statistics deterministically with respect to progress of primitives through the pipeline (and without requiring the pipeline to be flushed.) The PIPE_CONTROL command (see details below) is used to perform all of above synchronizations.

Top-of-Pipe Synchronization

Top-of-pipe synchronization refers to SW actions to prepare HW for new state-binding at the beginning of the rendering sequence in a given context. HW may have residual states cached in the state-caches and read-only surfaces in various caches. With new rendering sequence, read-only surfaces may go through change in the binding. Hence read-only invalidation is required before such new rendering sequence. Read-only cache invalidation is top-of-pipe synchronization. Upon parsing this specific pipe-control command, HW invalidates all caches in GT domain that have read-only surfaces but does not guarantee invalidation beyond GT caches . Further, HW does not guarantee that all prior accesses to those read-only surfaces have completed. Therefore SW must guarantee that there are no pending accesses to those read-only surfaces before initializing the top-of-pipe synchronization. PIPE-CONTROL command described below allows for invalidating individual read-only stream type. It is recommended that driver invalidates only the required caches on the need basis so that cache warm-up overhead can be reduced.

End-of-Pipe Synchronization

The driver can use end-of-pipe synchronization to know that rendering is complete (although not necessarily in memory) so that it can deallocate in-memory rendering state, read-only surfaces, instructions, and constant buffers. An end-of-pipe synchronization point is also sufficient to guarantee that all pending depth tests have completed so that the visible pixel count is complete prior to storing it to memory. End-of-pipe completion is sufficient (although not necessary) to guarantee that read events are complete (a “read fence” completion). Read events are still pending if work in the pipeline requires any type of read except a render target read (blend) to complete.

Write synchronization is a special case of end-of-pipe synchronization that requires that the render cache and/or depth related caches are flushed to memory, where the data will become globally visible. This type of synchronization is required prior to SW (CPU) actually reading the result data from memory, or initiating an operation that will use as a read surface (such as a texture surface) a previous render target and/or depth/stencil buffer. Exercising the write cache flush bits (Render Target Cache Flush Enable, Depth Cache Flush Enable, DC Flush) in PIPE_CONTROL only ensures the write caches are flushed and doesn’t guarantee the data is globally visible.

SW can track the completion of the end-of-pipe-synchronization by using “Notify Enable” and “Post-Sync Operation - Write Immediate Data” in the PIPE_CONTROL command. “Notify Enable” and “Post-

Sync Operation - Write Immediate Data" generate a fence cycle on achieving end-of-pipe-synchronization for the corresponding PIPE_CONTROL command. Fence cycle ensures all the write cycles in front of it are to global visible point before they themselves get processed.It is guaranteed the data flushed out by the PIPE_CONTROL is updated in memory by the time SW receives the corresponding Pipe Control Notify interrupt.

In case the data flushed out by the render engine is to be read back in to the render engine in coherent manner, then the render engine has to wait for the fence completion before accessing the flushed data. This can be achieved by following means on various products:

PIPE_CONTROL command with CS Stall and the required write caches flushed with Post-Sync-Operation as Write Immediate Data. Example: • WorkLoad-1 (3D/GPGPU/MEDIA) • PIPE_CONTROL (CS Stall, Post-Sync-Operation Write Immediate Data, Required Write Cache Flush bits set) • WorkLoad-2 (Can use the data produced or output by Workload-1)

Synchronization Actions

In order for the driver to act based on a synchronization point (usually the whole point), the reaching of the synchronization point must be communicated to the driver. This section describes the actions that may be taken upon completion of a synchronization point which can achieve this communication.

Writing a Value to Memory

The most common action to perform upon reaching a synchronization point is to write a value out to memory. An immediate value (included with the synchronization command) may be written. In lieu of an immediate value, the 64-bit value of the PS_DEPTH_COUNT (visible pixel count) or TIMESTAMP register may be written out to memory. The captured value will be the value at the moment all primitives parsed prior to the synchronization commands have been completely rendered, and optionally after all said primitives have been pushed to memory. It is not required that a value be written to memory by the synchronization command.

Visible pixel or TIMESTAMP information is only useful as a delta between 2 values, because these counters are free-running and are not to be reset except at initialization. To obtain the delta, two PIPE_CONTROL commands should be initiated with the command sequence to be measured between them. The resulting pair of values in memory can then be subtracted to obtain a meaningful statistic about the command sequence.

PS_DEPTH_COUNT

If the selected operation is to write the visible pixel count (PS_DEPTH_COUNT register), the synchronization command should include the **Depth Stall Enable** parameter. There is more than one point at which the global visible pixel count can be affected by the pipeline; once the synchronization command reaches the first point at which the count can be affected, any primitives following it are stalled at that point in the pipeline. This prevents the subsequent primitives from affecting the visible pixel count until all primitives preceding the synchronization point reach the end of the pipeline, the visible pixel count is accurate and the synchronization is completed. This stall has a minor effect on performance and should only be used in order to obtain accurate "visible pixel" counts for a sequence of primitives.

The PS_DEPTH_COUNT count can be used to implement an (API/DDI) "Occlusion Query" function.

Generating an Interrupt

The synchronization command may indicate that a "Sync Completion" interrupt is to be generated (if enabled by the MI Interrupt Control Registers – see *Memory Interface Registers*) once the rendering of all prior primitives is complete. Again, the completion of rendering can be considered to be when the internal render cache has been updated, or when the cache contents are visible in memory, as selected by the command options.

Invalidating of Caches

If software wishes to use the notification that a synchronization point has been reached in order to reuse referenced structures (surfaces, state, or instructions), it is not sufficient just to make sure rendering is complete. If additional primitives are initiated after new data is laid over the top of old in memory following a synchronization point, it is possible that stale cached data will be referenced for the subsequent rendering operation. In order to avoid this, the PIPE_CONTROL command must be used. (See PIPE_CONTROL Command description).

PIPE_CONTROL Command

The PIPE_CONTROL command is used to effect the synchronization described above. Parsing a PIPE_CONTROL command stalls the 3D pipe only if the stall enable bit is set. Commands after PIPE_CONTROL will continue to be parsed and processed in the 3D pipeline. This may include additional PIPE_CONTROL commands. The implementation does enforce a practical upper limit (8) on the number of PIPE_CONTROL commands that may be outstanding at once. Parsing a PIPE_CONTROL command that causes this limit to be reached will stall the parsing of new commands until the first of the outstanding PIPE_CONTROL commands reaches the end of the pipe and retires.

Although PIPE_CONTROL is intended for use with the 3D pipe, it is legal to issue PIPE_CONTROL when the Media pipe is selected. In this case PIPE_CONTROL will stall at the top of the pipe until the Media FFs finish processing commands parsed before PIPE_CONTROL. Post-synchronization operations, flushing of caches and interrupts will then occur if enabled via PIPE_CONTROL parameters. Due to this stalling behavior, only one PIPE_CONTROL command can be outstanding at a time on the Media pipe.

For the invalidate operation of the pipe control, the following pointers are affected. The invalidate operation affects the restore of these packets. If the pipe control invalidate operation is completed before the context save, the indirect pointers will not be restored from memory.

- Pipeline State Pointer
- Media State Pointer
- Constant Buffer Packet

It is up to software to program the appropriate read-only cache invalidation such as the sampler and constant read caches or the instruction and state caches. Once notification is observed, new data may then be loaded (potentially "on top of" the old data) without fear of stale cache data being referenced for subsequent rendering.

If software wishes to access the rendered data in memory (for analysis by the application or to copy it to a new location to use as a texture, for example), it must also ensure that the write cache (render cache) is flushed after the synchronization point is reached so that memory will be updated. This can be done by setting the **Write Cache Flush Enable** bit. Note that the **Depth Stall Enable** bit must be clear in order for the flush of the render cache to occur. **Depth Stall Enable** is intended only for accurate reporting of the PS_DEPTH counter; the render cache cannot be flushed nor can the read caches be invalidated (except for the instruction/state cache) in conjunction with this operation.

Vertex caches are only invalidated when the VF invalidate bit is set in PIPE_CONTROL (i.e. decision is done in software, not hardware) Note that the index-based vertex cache is always flushed between primitive topologies and of course PIPE_CONTROL can only be issued between primitive topologies. Therefore only the VF ("address-based") cache is uniquely affected by PIPE_CONTROL.

PIPE_CONTROL

PIPE_CONTROL

Description
Hardware supports up to 32 pending PIPE_CONTROL flushes.

The table below explains all the different flush/invalidation scenarios.

Caches Invalidated/Flushed by PIPE_CONTROL Bit Settings

Write Cache Flush	Notification Enabled	Non-VF RO Cache Invalidate	VF RO Cache Invalidate	Marker Sent	Pipeline Marker Enable	Completion Requested	Top of Pipe Invalidate Pulse from CS
0	0	0	0	N/A	N/A	N/A	N/A
0	0	0	1	Yes	No	N/A	No
0	0	1	0	No	N/A	N/A	Yes
0	0	1	1	Yes	No	No	Yes
X	1	0	X	Yes	Yes	Yes	No
X	1	1	X	Yes	Yes	Yes	Yes
1	X	0	X	Yes	Yes	Yes	No
1	X	1	X	Yes	Yes	Yes	Yes

Programming Restrictions for PIPE_CONTROL

PIPE_CONTROL arguments can be split up into three categories:

- Post-sync operations
- Flush Types
- Stall

Post-sync operation is only indirectly affected by the flush type category via the stall bit. The stall category depends on the both flush type and post-sync operation arguments. A PIPE_CONTROL with no arguments set is **Invalid**.

Post-Sync Operation

These arguments relate to events that occur after the marker initiated by the PIPE_CONTROL command is completed. The table below shows the restrictions:

Argument	Bits	Restriction
LRI Post Sync Operation	23	Post Sync Operation ([15:14] of DW1) must be set to 0x0.
LRI Post Sync Operation	23	Requires stall bit ([20] of DW1) set.
LRI Post Sync Operation	23	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Global Snapshot Count Reset	19	This bit must not be exercised on any product. Requires stall bit ([20] of DW1) set.
Generic Media State Clear	16	Requires stall bit ([20] of DW1) set.
Indirect State Pointers Disable	9	Requires stall bit ([20] of DW1) set.
Store Data Index	21	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0'.
Sync GFDT	17	Post-Sync Operation ([15:14] of DW1) must be set to something other than '0' or 0x2520[13] must be set.
TLB inv	18	Requires stall bit ([20] of DW1) set.
Post Sync Op	15:14	LRI Post Sync Operation ([23] of DW1) must be set to '0'.
Post Sync Op	15:14	"CS Stall" must always be set in all PIPE_CONTROL commands having "Post-Sync Operation" set to "Report PS Depth Count" or "Report Time Stamp" when user wishes to set "Post-Sync Operation" bit in any of the MI_ATOMIC or MI_SEMAPHORE_SIGNAL commands programmed.
Post Sync Op	15:14	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Notify En	8	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.

Flush Types

These are arguments related to the type of read only invalidation or write cache flushing is being requested. Note that there is only intra-dependency. That is, it is not affected by the post-sync operation or the stall bit. The table below shows the restrictions:

Arguments	Bit	Restrictions
Depth Stall	13	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Render Target Cache Flush	12	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Depth Cache Flush	0	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Stall Pixel Scoreboard	1	No Restriction.
DC Flush Enable	5	Requires stall bit ([20] of DW) set for all GPGPU and Media Workloads.
Inst invalidate	11	No Restriction.
Tex invalidate	10	No Restriction.
VF invalidate	4	"Post Sync Operation" must be enabled to "Write Immediate Data" or "Write PS Depth Count" or "Write Timestamp".
Constant invalidate	3	No Restriction.
State Invalidate	2	No Restriction.

Stall

If the stall bit is set, the command streamer waits until the pipe is completely flushed.

Arguments	Bit	Restrictions
Stall Bit	20	[All Stepping][All SKUs]: One of the following must also be set: • Render Target Cache Flush Enable ([12] of DW1) • Depth Cache Flush Enable ([0] of DW1) • Stall at Pixel Scoreboard ([1] of DW1) • Depth Stall ([13] of DW1) • Post-Sync Operation ([13] of DW1) • DC Flush Enable ([5] of DW1)

Render Logical Context Data

Logical Contexts are memory images used to store copies of the device's rendering and ring context.

Logical Contexts are aligned to 256-byte boundaries.

Logical contexts are referenced by their memory address. The format and contents of rendering contexts are considered ***device-dependent*** and software must not access the memory contents directly. The definition of the logical rendering and power context memory formats is included here primarily for internal documentation purposes.

Overall Context Layout

Content for this topic is currently under development.

Context Layout

For [CHV, BSW], when Execlists are enabled, the Context Image for the rendering engine consists of 20 4K pages:

Per-Process HW Status Page (4K)
Register State Context

When Execlists are disabled, the context image doesn't consist the Per-Process HW status page.

Register State context is explained in detail in "Register State Context" Section.

Register State Context

This section describes programming requirements for the Register State Context.

Programming Note	
Context:	Register State Context.

- All the MMIO registers part of the "Register State Context Image" are context specific and gets context save/restored upon a context switch. MMIO register values belonging to a context can be exercised through HOST/IA MMIO interface only when the context is active in HW. Exercising context specific MMIO registers through HOST/IA MMIO is completely asynchronous to the context execution in HW and can't guarantee a desired sampling point during execution. In execlist mode of scheduling there is no active context when HW is Idle.
- All the write access to MMIO registers listed in the "Register State Context image" subsections below must be done through MI commands (MI_LOAD_REGISTER_IMM, MI_LOAD_REG_MEM, MI_LOAD_REGISTER_REG) in the command sequence.
- MMIO reads or writes to any of the registers listed in the "Register State Context image" subsections through HOST/IA MMIO interface must follow the steps below:
  - SW should set the Force Wakeup bit to prevent GT from entering C6.
  - Write 0x2050[31:0] = 0x00010001 (disable sequence).
  - Disable IDLE messaging in CS (Write 0x2050[31:0] = 0x00010001).
  - Poll/Wait for register bits of 0x22AC[6:0] turn to 0x30 value.
  - Read/Write to desired MMIO registers.
  - Enable IDLE messaging in CS (Write 0x2050[31:0] = 0x00010000).
  - Force Wakeup bit should be reset to enable C6 entry.

Register State Context

Context Color Codes Used

EXECLIST CONTEXT
EXECLIST CONTEXT (PPGTT Base)
ENGINE CONTEXT
EXTENDED ENGINE CONTEXT
URB_ATOMIC CONTEXT

Register Information

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
NOOP		CS	1	0
Load_Register_Immediate header	0x1100_1015	CS	1	0001
Context Control	0x2244	CS	2	0002
Ring Head Pointer Register	0x2034	CS	2	0004
Ring Tail Pointer Register	0x2030	CS	2	0006
RING_BUFFER_START	0x2038	CS	2	0008
RING_BUFFER_CONTROL	0x203C	CS	2	000A
Batch Buffer Current Head Register (UDW)	0x2168	CS	2	000C
Batch Buffer Current Head Register	0x2140	CS	2	000E
Batch Buffer State Register	0x2110	CS	2	0010
SECOND_BB_ADDR_UDW	0x211C	CS	2	0012
SECOND_BB_ADDR	0x2114	CS	2	0014
SECOND_BB_STATE	0x2118	CS	2	0016
NOOP		CS	8	0018
NOOP		CS	1	0020
Load_Register_Immediate header	0x1100_1011	CS	1	0021
CTX_TIMESTAMP	23A8	CS	2	0022
PDP3_UDW	228C	CS	2	0024
PDP3_LDW	2288	CS	2	0026
PDP2_UDW	2284	CS	2	0028
PDP2_LDW	2280	CS	2	002A
PDP1_UDW	227C	CS	2	002C
PDP1_LDW	2278	CS	2	002E
PDP0_UDW	2274	CS	2	0030
PDP0_LDW	2270	CS	2	0032
NOOP		CS	12	0034

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
NOOP		CS	1	0040
Load_Register_Immediate header	0x1100_0001	CS	1	0041
R_PWR_CLK_STATE	20C8	CS	2	0042
GPGPU_CSR_BASE_ADDRESS		CS	3	0044
NOOP		CS	9	0047
NOOP		CS	5	0050
Load_Register_Immediate header	0x1100_10D5	CS	1	0055
EXCC	0x2028	CS	2	0056
MI_MODE	0x209C	CS	2	0058
INSTPM	0x20C0	CS	2	005A
PR_CTR_CTL	0x2178	CS	2	005C
PR_CTR_THRSH	0x217C	CS	2	005E
IA_VERTICES_COUNT	0x2310	CS	4	0062
IA_PRIMITIVES_COUNT	0x2318	CS	4	0066
VS_INVOCATION_COUNT	0x2320	CS	4	006A
HS_INVOCATION_COUNT	0x2300	CS	4	006E
DS_INVOCATION_COUNT	0x2308	CS	4	0072
GS_INVOCATION_COUNT	0x2328	CS	4	0076
GS_PRIMITIVES_COUNT	0x2330	CS	4	007A
CL_INVOCATION_COUNT	0x2338	CS	4	007E
CL_PRIMITIVES_COUNT	0x2340	CS	4	0082
PS_INVOCATION_COUNT_0	0x22C8	CS	4	0086
PS_DEPTH_COUNT _0	0x22D8	CS	4	008A
TIMESTAMP Register (LSB)	0x2358	CS	2	008E
CS_INTERNAL	0x2580	CS	2	0090
GPUGPU_DISPATCHDIMX	0x2500	CS	2	0092
GPUGPU_DISPATCHDIMY	0x2504	CS	2	0094
GPUGPU_DISPATCHDIMZ	0x2508	CS	2	0096
MI_PREDICATE_SRC0	0x2400	CS	2	0098
MI_PREDICATE_SRC0	0x2404	CS	2	009A
MI_PREDICATE_SRC1	0x2408	CS	2	009C
MI_PREDICATE_SRC1	0x240C	CS	2	009E
MI_PREDICATE_DATA	0x2410	CS	2	00A0
MI_PREDICATE_DATA	0x2414	CS	2	00A2
MI_PRED_RESULT	0x2418	CS	2	00A4
3DPRIM_END_OFFSET	0x2420	CS	2	00A6

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
3DPRIM_START_VERTEX	0x2430	CS	2	00A8
3DPRIM_VERTEX_COUNT	0x2434	CS	2	00AA
3DPRIM_INSTANCE_COUNT	0x2438	CS	2	00AC
3DPRIM_START_INSTANCE	0x243C	CS	2	00AE
3DPRIM_BASE_VERTEX	0x2440	CS	2	00B0
GPGPU_THREADS_DISPATCHED	0x2290	CS	4	00B2
PS_INVOCATION_COUNT_1	0x22F0	CS	4	00B6
PS_DEPTH_COUNT _1	0x22F8	CS	4	00BA
BB_START_ADDR_UDW	0x2170	CS	2	00BE
BB_START_ADDR	0x2150	CS	2	00C0
BB_ADD_DIFF	0x2154	CS	2	00C2
BB_OFFSET	0x2158	CS	2	00C4
RS_PREEMPT_STATUS	0x215C	CS	2	00C6
CTX_SEMA_REG	0x24B4	CS	4	00C8
PRODUCE_COUNT_BTP	0x2480	CS	2	00CC
PRODUCE_COUNT_DX9_CONSTANTS	0x2484	CS	2	00CE
PRODUCE_COUNT_GATHER_CONSTANTS	0x248C	CS	2	00D0
PARSED_COUNT_BTP	0x2490	CS	2	00D2
PARSED_COUNT_DX9_CONSTANTS	0x2494	CS	2	00D4
MI_PREDICATE_RESULT_1	0x241C	CS	2	00D6
CS_GPR (1-16)	0x2600	CS	64	00D8
MI_TAGDATA	0x219C	CS	4	011C
PS_INVOCATION_COUNT_2	0x2348	CS	4	0124
PS_DEPTH_COUNT_2	0x2350	CS	4	0128
NOOP		CS	12	012C
MI_TOPOLOGY_FILTER		CS	1	0138
MI_URB_CLEAR		CS	2	0139
PIPELINE_SELECT		CS	1	013B
STATE_BASE_ADDRESS		CS	16	013C
3DSTATE_PUSH_CONSTANT_ALLOC_VS		CS	2	014C
3DSTATE_PUSH_CONSTANT_ALLOC_HS		CS	2	014E
3DSTATE_PUSH_CONSTANT_ALLOC_DS		CS	2	0150
3DSTATE_PUSH_CONSTANT_ALLOC_GS		CS	2	0152
3DSTATE_PUSH_CONSTANT_ALLOC_PS		CS	2	0154
3DSTATE_BINDING_TABLE_POOL_ALLOC		CS	4	0156
3DSTATE_GATHER_POOL_ALLOC		CS	4	015A

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC		CS	4	015E
MI_RS_CONTROL		CS	1	0162
MI_URB_ATOMIC_ALLOC		CS	1	0163
NOOP		CS	12	0164
3DSTATE_VS		SVG	9	0170
3DSTATE_BINDING_TABLE_POINTERS_VS		SVG	2	0179
3DSTATE_SAMPLER_STATE_POINTERS_VS		SVG	2	017B
3DSTATE_URB_VS		SVG	2	017D
3DSTATE_HS		SVG	9	017F
3DSTATE_BINDING_TABLE_POINTERS_HS		SVG	2	0188
3DSTATE_SAMPLER_STATE_POINTERS_HS		SVG	2	018A
3DSTATE_URB_HS		SVG	2	018C
3DSTATE_TE		SVG	4	018E
3DSTATE_DS		SVG	9	0192
3DSTATE_BINDING_TABLE_POINTERS_DS		SVG	2	019B
3DSTATE_SAMPLER_STATE_POINTERS_DS		SVG	2	019D
3DSTATE_URB_DS		SVG	2	019F
3DSTATE_GS		SVG	10	01A1
3DSTATE_BINDING_TABLE_POINTERS_GS		SVG	2	01AB
3DSTATE_SAMPLER_STATE_POINTERS_GS		SVG	2	01AD
3DSTATE_URB_GS		SVG	2	01AF
3DSTATE_STREAMOUT		SVG	5	01B1
3DSTATE_CLIP		SVG	4	01B6
3DSTATE_SF		SVG	4	01BA
3DSTATE_SCISSOR_STATE_POINTERS		SVG	2	01BE
3DSTATE_VIEWPORT_STATE_POINTERS_CL_SF		SVG	2	01C0
3DSTATE_RASTER		SVG	5	01C2
3DSTATE_WM_HZ_OP		SVG	5	01C7
3DSTATE_MULTISAMPLE		SVG	2	01CC
3DSTATE_CONSTANT_VS		SVG	11	01CE
3DSTATE_CONSTANT_HS		SVG	11	01D9
3DSTATE_CONSTANT_DS		SVG	11	01E4
3DSTATE_CONSTANT_GS		SVG	11	01EF
3DSTATE_DRAW_RECTANGULAR		SVG	4	01FA
3DSTATE_SW_TE_BASE_ADDRESS		SVG	3	01FE
Load_Register_Immediate header	0x1100_1001	SVG	1	0201

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
FF_PERF_REG	0x6b1c	SVG	2	0202
NOOP		SVG	12	0204
3DSTATE_WM		SVL	2	0210
3DSTATE_VIEWPORT_STATE_POINTER_CC		SVL	2	0212
3DSTATE_CC_STATE_POINTERS		SVL	2	0214
3DSATE_WM_SAMPLEMASK		SVL	2	0216
3DSTATE_DEPTH_STENCIL		SVL	3	0218
3DSTATE_WM_CHROMAKEY		SVL	2	021B
3DSTATE_DEPTH_BUFF		SVL	8	021D
3DSTATE_HIZ_DEPTH_BUFF		SVL	5	0225
3DSTATE_STC_DEPTH_BUFF		SVL	5	022A
3DSTATE_CLEAR_PARAMS		SVL	3	022F
3DSTATE_SBE		SVL	4	0232
3DSTATE_SBE_SWIZ		SVL	11	0236
3DSTATE_PS		SVL	12	0241
3DSTATE_BINDING_TABLE_POINTERS_PS		SVL	2	024D
STATE_SAMPLER_STATE_POINTERS_PS		SVL	2	024F
3DSTATE_BLEND_STATE_POINTERS		SVL	2	0251
3DSTATE_PS_EXTRA		SVL	2	0253
3DSTATE_PS_BLEND		SVL	2	0255
3DSTATE_CONSTANT_PS		SVL	11	0257
STATE_SIP		SVL	3	0262
3DSTATE_SAMPLE_PATTERN		SVL	9	0265
Load_Register_Immediate header	0x1100_1021	SVL	1	026E
Cache_Mode_0	0x7000	SVL	2	026F
Cache_Mode_1	0x7004	SVL	2	0271
GT_MODE	0x7008	SVL	2	0273
FBC_RT_BASE_ADDR_REGISTER	0x7020	SVL	2	0279
FBC_RT_BASE_ADDR_REGISTER_UPPER	0x7024	SVL	2	027B
SAMPLER_MODE	0x7028	SVL	2	027D
L3_Config	0x7034	SVL	2	0287
GLOBAL_CLEAR_VALUE_0	0x7700	SVL	2	0289
GLOBAL_CLEAR_VALUE_1	0x7704	SVL	2	028B
GLOBAL_CLEAR_VALUE_2	0x7708	SVL	2	028D
GLOBAL_CLEAR_VALUE_3	0x770C	SVL	2	028F
NOOP		SVL	15	0291

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
NOOP		TDL	1	02A0
Load_Register_Immediate header	0x1100_103B	TDL	1	02A1
TD_CTL	E400	TDL	2	02A2
TD_CTL2	E404	TDL	2	02A4
TD_VF_VS_EMSK	E408	TDL	2	02A6
TD_GS_EMSK	E40C	TDL	2	02A8
TD_WIZ_EMSK	E410	TDL	2	02AA
TD_TS_EMSK	E428	TDL	2	02AC
TD_HS_EMSK	E4B0	TDL	2	02AE
TD_DS_EMSK	E4B4	TDL	2	02B0
EU_PERF_CNT_CTL0	E458	TDL	2	02CE
EU_PERF_CNT_CTL1	E558	TDL	2	02D0
EU_PERF_CNT_CTL2	E658	TDL	2	02D2
EU_PERF_CNT_CTL3	E758	TDL	2	02D4
EU_PERF_CNT_CTL4	E45C	TDL	2	02D6
EU_PERF_CNT_CTL5	E55C	TDL	2	02D8
EU_PERF_CNT_CTL6	E65C	TDL	2	02DA
NOOP		TDL	2	02DE
NOOP		WM	1	02E0
Load_Register_Immediate header	0x1100_1005	WM	1	02E1
WMHWCLRVAL	0x5524	WM	2	02E6
3DSTATE_POLY_STIPPLE_PATTERN		WM	33	02E8
3DSTATE_AA_LINE_PARAMS		WM	3	0309
3DSTATE_POLY_STIPPLE_OFFSET		WM	2	030C
3DSTATE_LINE_STIPPLE		WM	3	030E
NOOP		WM	15	0311
NOOP		SC	1	0320
Load_Register_Immediate header	0x1100_1005	SC	1	0321
3DSTATE_MONOFILTER_SIZE		SC	2	0328
3DSTATE_CHROMA_KEY		SC	16	032A
NOOP		SC	6	033A
NOOP		DM	1	0340
3DSTATE_SAMPLER_PALETTE_LOAD0		DM	257	0341
NOOP		DM	1	0442
3DSTATE_SAMPLER_PALETTE_LOAD1		DM	257	0443
NOOP		DM	12	0544

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
NOOP		SOL	1	0550
Load_Register_Immediate header	0x1100_1027	SOL	1	0551
SO_NUM_PRIMS_WRITTEN0	0x5200	SOL	4	0552
SO_NUM_PRIMS_WRITTEN1	0x5208	SOL	4	0556
SO_NUM_PRIMS_WRITTEN2	0x5210	SOL	4	055A
SO_NUM_PRIMS_WRITTEN3	0x5218	SOL	4	055E
SO_PRIM_STORAGE_NEEDED0	0x5240	SOL	4	0562
SO_PRIM_STORAGE_NEEDED1	0x5248	SOL	4	0566
SO_PRIM_STORAGE_NEEDED2	0x5250	SOL	4	056A
SO_PRIM_STORAGE_NEEDED3	0x5258	SOL	4	056E
SO_WRITE_OFFSET0	0x5280	SOL	2	0572
SO_WRITE_OFFSET1	0x5284	SOL	2	0574
SO_WRITE_OFFSET2	0x5288	SOL	2	0576
SO_WRITE_OFFSET3	0x528C	SOL	2	0578
3DSTATE_SO_BUFFER		SOL	32	057A
NOOP		SOL	1	059A
3DSTATE_SO_DECL_LIST		SOL	259	059B
NOOP		SOL	2	069E
3DSTATE_INDEX_BUFFER		VF	5	06A0
3DSTATE_VERTEX_BUFFERS		VF	133	06A5
3DSTATE_VERTEX_ELEMENTS		VF	69	072A
3DSTATE_VF_STATISTICS		VF	1	076F
3DSTATE_VF		VF	2	0770
3DSTATE_SGVS		VF	2	0772
3DSTATE_VF_INSTANCING		VF	69	0774
3DSTATE_VF_TOPOLOGY		VF	2	07B9
Load_Register_Immediate header	0x1100_1095	VF	1	07BB
INSTANCE CNT	08300 - 08384h	VF	68	07BC
INSTANCE INDX	08400 - 08484h	VF	68	0800
COMMITTED VERTEX NUMBER	08390h	VF	2	0844
COMMITTED INSTANCE ID	08394h	VF	2	0846
COMMITTED PRIMITIVE ID	08398h	VF	2	0848
STATUS	0839Ch	VF	2	084A
COMMON VERTEX	083A0h	VF	2	084C
VF_GUARDBAND	083A4h	VF	2	0852
NOOP		VF	14	0852

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
Load_Register_Immediate header	0x1100_105D	VFE	1	0860
TDL0 DATA		VFE	94	0861
NOOP		VFE	1	08BF
Load_Register_Immediate header	0x1100_105D	VFE	1	08C0
TDL1 DATA		VFE	94	08C1
NOOP		VFE	1	091F
Load_Register_Immediate header	0x1100_105D	VFE	1	0920
TDL2 DATA		VFE	94	0921
NOOP		VFE	1	097F
Load_Register_Immediate header	0x1100_105D	VFE	1	0980
TDL3 DATA		VFE	94	0981
NOOP		VFE	1	09DF
Load_Register_Immediate header	0x1100_105D	VFE	1	09E0
TDL4 DATA		VFE	94	09E1
NOOP		VFE	1	0A3F
Load_Register_Immediate header	0x1100_105D	VFE	1	0A40
TDL5 DATA		VFE	94	0A41
NOOP		VFE	1	0A9F
Load_Register_Immediate header	0x1100_105D	VFE	1	0AA0
TDL6 DATA		VFE	94	0AA1
NOOP		VFE	1	0AFF
Load_Register_Immediate header	0x1100_105D	VFE	1	0B00
TDL7 DATA		VFE	94	0B01
NOOP		VFE	1	0B5F
Load_Register_Immediate header	0x1100_105D	VFE	1	0B60
TDL8 DATA		VFE	94	0B61
NOOP		VFE	1	0BBF
Load_Register_Immediate header	0x1100_1023	VFE	1	0BC0
GW0 DATA		VFE	36	0BC1
NOOP		VFE	11	0BE5
Load_Register_Immediate header	0x1100_1023	VFE	1	0BF0
GW1 DATA		VFE	36	0BF1
NOOP		VFE	11	0C15
Load_Register_Immediate header	0x1100_1023	VFE	1	0C20
GW2 DATA		VFE	36	0C21
NOOP		VFE	11	0C45

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
Load_Register_Immediate header	0x1100_1023	VFE	1	0C50
GW3 DATA		VFE	36	0C51
NOOP		VFE	11	0C75
Load_Register_Immediate header	0x1100_1023	VFE	1	0C80
GW4 DATA		VFE	36	0C81
NOOP		VFE	11	0CA5
Load_Register_Immediate header	0x1100_1023	VFE	1	0CB0
GW5 DATA		VFE	36	0CB1
NOOP		VFE	11	0CD5
Load_Register_Immediate header	0x1100_1023	VFE	1	0CE0
GW6 DATA		VFE	36	0CE1
NOOP		VFE	11	0D05
Load_Register_Immediate header	0x1100_1023	VFE	1	0D10
GW7 DATA		VFE	36	0D11
NOOP		VFE	11	0D35
Load_Register_Immediate header	0x1100_1023	VFE	1	0D40
GW8 DATA		VFE	36	0D41
NOOP		VFE	11	0D65
Load_Register_Immediate header	0x1100_1017	VFE	1	0D70
TSG0 DATA		VFE	24	0D71
NOOP		VFE	7	0D89
Load_Register_Immediate header	0x1100_1017	VFE	1	0D90
TSG1 DATA		VFE	24	0D91
NOOP		VFE	7	0DA9
Load_Register_Immediate header	0x1100_1017	VFE	1	0DB0
TSG2 DATA		VFE	24	0DB1
NOOP		VFE	7	0DC9
Load_Register_Immediate header	0x1100_1017	VFE	1	0DD0
TSG3 DATA		VFE	24	0DD1
NOOP		VFE	7	0DE9
Load_Register_Immediate header	0x1100_1017	VFE	1	0DF0
TSG4 DATA		VFE	24	0DF1
NOOP		VFE	7	0E09
Load_Register_Immediate header	0x1100_1017	VFE	1	0E10
TSG5 DATA		VFE	24	0E11
NOOP		VFE	7	0E29

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
Load_Register_Immediate header	0x1100_1017	VFE	1	0E30
TSG6 DATA		VFE	24	0E31
NOOP		VFE	7	0E49
Load_Register_Immediate header	0x1100_1017	VFE	1	0E50
TSG7 DATA		VFE	24	0E51
NOOP		VFE	7	0E69
Load_Register_Immediate header	0x1100_1017	VFE	1	0E70
TSG8 DATA		VFE	24	0E71
NOOP		VFE	7	0E89
Load_Register_Immediate header	0x1100_1009	VFE	1	0E90
VFE Data		VFE	10	0E91
NOOP		VFE	5	0E9B
MEDIA_VFE_STATE		VFE	9	0EA0
MEDIA_CURBE_LOAD		VFE	4	0EA9
MEDIA_INTERFACE_DESCRIPTOR_LOAD		VFE	4	0EAD
NOOP		VFE	15	0EB1
NOOP		RS	2	0EC0
3DSTATE_BINDING_TABLE_POOL_ALLOC		RS	4	0EC2
3DSTATE_BINDING_TABLE_EDIT_VS		RS	258	0EC6
NOOP		RS	6	0FC8
3DSTATE_BINDING_TABLE_EDIT_GS		RS	258	0FCE
NOOP		RS	6	10D0
3DSTATE_BINDING_TABLE_EDIT_HS		RS	258	10D6
NOOP		RS	6	11D8
3DSTATE_BINDING_TABLE_EDIT_DS		RS	258	11DE
NOOP		RS	6	12E0
3DSTATE_BINDING_TABLE_EDIT_PS		RS	258	12E6
3DSTATE_GATHER_POOL_ALLOC		RS	4	13E8
MI_BATCH_BUFFER_END/NOOP ***		RS	1	13EC
NOOP		RS	5	13ED
3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC		RS	4	13F2
3DSTATE_DX9_CONSTANTF_VS(Global)		RS	1026	13F6
NOOP		RS	6	17F8
3DSTATE_DX9_CONSTANTI_VS(Global)		RS	130	17FE
NOOP		RS	6	1880
3DSTATE_DX9_CONSTANTB_VS(Global)		RS	18	1886

Description	MMIO Offset/Command	Unit	# of DW	Address Offset
NOOP		RS	6	1898
3DSTATE_DX9_CONSTANTF_VS(local)		RS	1026	189E
NOOP		RS	6	1CA0
3DSTATE_DX9_CONSTANTI_VS(local)		RS	130	1CA6
NOOP		RS	6	1D28
3DSTATE_DX9_CONSTANTB_VS(local)		RS	18	1D2E
NOOP		RS	3	1D40
3DSTATE_DX9_LOCAL_VALID_VS		RS	11	1D43
3DSTATE_DX9_CONSTANTF_PS(Global)		RS	1026	1D4E
NOOP		RS	6	2150
3DSTATE_DX9_CONSTANTI_PS(Global)		RS	130	2156
NOOP		RS	6	21D8
3DSTATE_DX9_CONSTANTB_PS(Global)		RS	18	21DE
NOOP		RS	6	21F0
3DSTATE_DX9_CONSTANTF_PS(local)		RS	1026	21F6
NOOP		RS	6	25F8
3DSTATE_DX9_CONSTANTI_PS(local)		RS	130	25FE
NOOP		RS	6	2680
3DSTATE_DX9_CONSTANTB_PS(local)		RS	18	2686
3DSTATE_DX9_LOCAL_VALID_PS		RS	11	2698
MI_BATCH_BUFFER_END		RS	1	26A3
NOOP		RS	12	26A4
URB_ATOMIC_STORAGE		GAFS	8192	26B0
				46B0
			DW	18096
			Kbytes	70.6875

Ring Buffer

Ring Buffer can exist anywhere in memory mapped via Global GTT. Ring buffer details are mentioned in the ring context area of LRCA (Ring Buffer - Start Address, Head Offset, Tail Pointer & Control Register) in Execution List mode of scheduling. Ring Buffer registers are directly programmed in Ring Buffer mode of scheduling.

Ring Context

Ring Context starts at 4K offset from LRCA. Ring context contains all the details that are needed to be initialized by SW for submitting a context to HW for execution (Ring Buffer Details, Page Directory Information ..etc). Ring context is five cachelines in size. Note that the last cacheline of the ring context is specific for a given Engine and hence SW needs to populate it accordingly.

Ring Context comprises of the EXECLIST CONTEXT, EXECLIST CONTEXT (PPGTT Base) of the register state context. In Ring Buffer mode of scheduling EXECLIST CONTEXT contents are save/restored as NOOPS by HW.

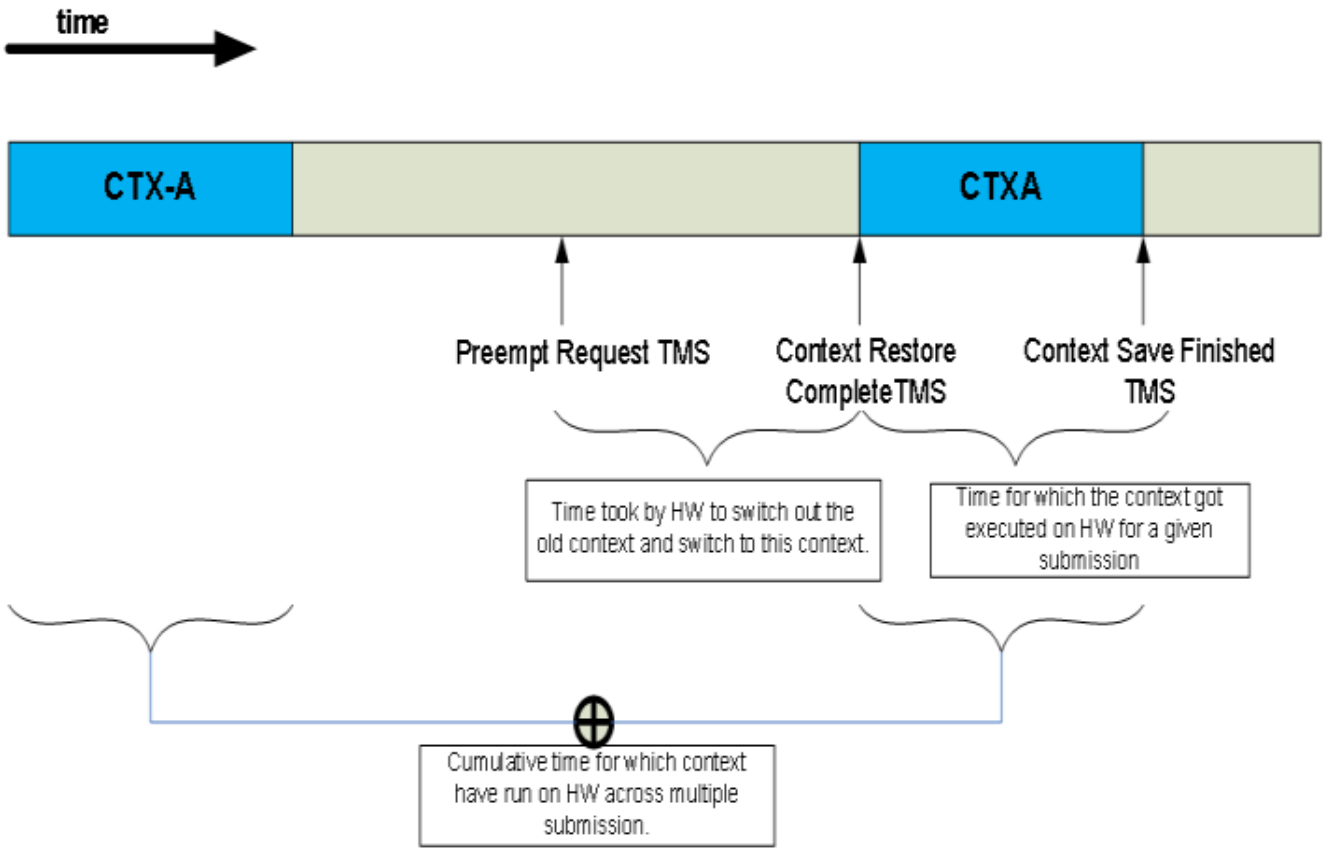
EXECLIST CONTEXT
EXECLIST CONTEXT(PPGTT Base)

The Per-Process Hardware Status Page

The layout of the Per-Process Hardware Status Page is defined at **PPHWSP_LAYOUT**.

The DWord offset values in the table are in decimal.

Figure below explains the different timestamp values reported to PPHWSP on a context switch.



This page is designed to be read by SW to glean additional details about a context beyond what it can get from the context status.

Accesses to this page are automatically treated as cacheable and snooped. It is therefore illegal to locate this page in any region where snooping is illegal (such as in stolen memory).

Render Engine Power Context

Table below captures the data from CS power context save/restored by PM. Address offset in the below table is relative to the starting location of CS in the overall power context image managed by PM.

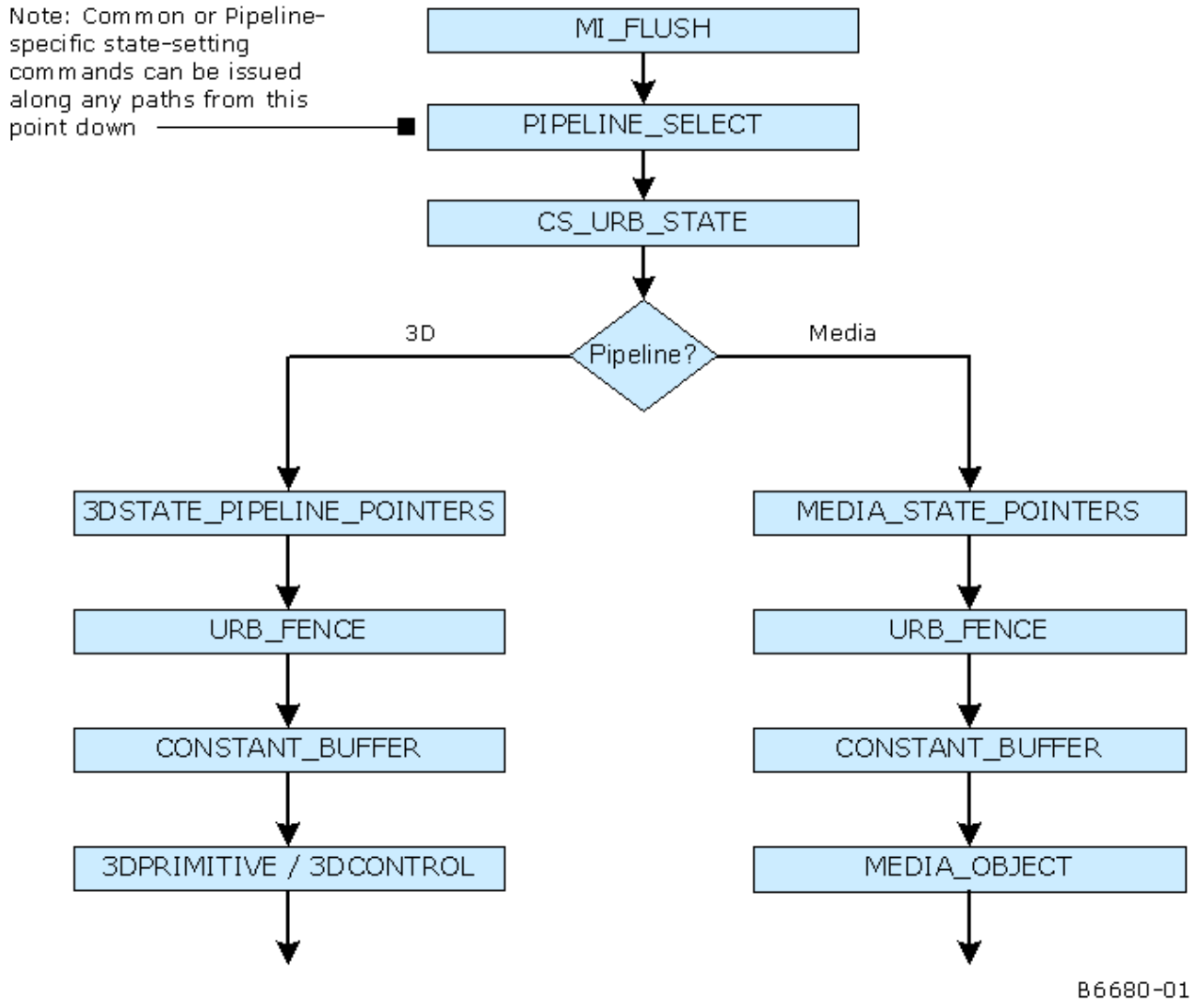
Description	Offset	Unit	# of DW	Address Offset (PWR)
NOOP		CS	1	0
Load_Register_Immediate header	0x1100_10A7	CS	1	0001
GFX_MODE	0x229C	CS	2	0002
GHWSP	0x2080	CS	2	0004
RING_BUFFER_CONTROL (Ring Always Disabled)	0x203C	CS	2	0006
Ring Head Pointer Register	0x2034	CS	2	0008
Ring Tail Pointer Register	0x2030	CS	2	000A
RING_BUFFER_START	0x2038	CS	2	000C
RING_BUFFER_CONTROL (Original status)	0x203C	CS	2	000E
Batch Buffer Current Head Register (UDW)	0x2168	CS	2	0010
Batch Buffer Current Head Register	0x2140	CS	2	0012
Batch Buffer State Register	0x2110	CS	2	0014
SECOND_BB_ADDR_UDW	0x211C	CS	2	0016
SECOND_BB_ADDR	0x2114	CS	2	0018
SECOND_BB_STATE	0x2118	CS	2	001A
RC_PSMI_CONTROL	0x2050	CS	2	001C
RC_PWRCTX_MAXCNT	0x2054	CS	2	001E
CTX_WA_PTR	0x2058	CS	2	0020
NOPID	0x2094	CS	2	0026
HWSTAM	0x2098	CS	2	0028
FF_THREAD_MODE	0x20A0	CS	2	002A
IMR	0x20A8	CS	2	002C
EIR (Restoring EIR doesn't it clear)	0x20B0	CS	2	002E
EMR	0x20B4	CS	2	0030
CMD_CCTL_0	0x20C4	CS	2	0032
GAFS_Mode	0x212C	CS	2	0038
UHPTR	0x2134	CS	2	003A
BB_PREEMPT_ADDR_UDW	0x216C	CS	2	003C
BB_PREEMPT_ADDR	0x2148	CS	2	003E
RING_BUFFER_HEAD_PREEMPT_REG	0x214C	CS	2	0040
CXT_SIZE	0x21A8	CS	2	0042
CXT_OFFSET	0x21AC	CS	2	0044
PREEMPT_DLY	0x2214	CS	2	0046

Description	Offset	Unit	# of DW	Address Offset (PWR)
MTCH_CID_RST (should not be required any more)	0x222C	CS	2	0048
SYNC_FLIP_STATUS	0x22D0	CS	2	004C
SYNC_FLIP_STATUS_1	0x22D4	CS	2	004E
WAIT_FOR_RC6_EXIT	0x20CC	CS	2	0054
RS_OFFSET	0x21B4	CS	2	0056
RCS_CTXID_PREEMPTION_HINT	0x24CC	CS	2	0058
RS_PREEMPTION_HINT_UDW	0x24C4	CS	2	005A
RS_PREEMPTION_HINT	0x24C0	CS	2	005C
CS_PREEMPTION_HINT_UDW	0x24C8	CS	2	005E
CS_PREEMPTION_HINT	0x24BC	CS	2	0060
CCID Register	0x2180	CS	2	0062
SBB_PREEMPT_ADDRESS_UDW	0x2138	CS	2	0064
SBB_PREEMPT_ADDRESS	0x213C	CS	2	0066
URB_CTX_OFFSET	0x21B8	CS	2	0068
MI_PREDICATE_RESULT_2	0x23BC	CS	2	006A
CTXT_ST_PTR	0x23A0	CS	2	006C
CTXT_ST_BUF	0x2370	CS	24	006E
SEMA_WAIT_POLL	0x224C	CS	2	0086
IDLEDELAY	0x223C	CS	2	0088
DISPLAY MESSAGE FORWARD STATUS	0x22E8	CS	2	008A
RCS_FORCE_TO_NONPRIV	0x24D0	CS	24	008C
VF PREMPTION VERTEX HINT	0x83B0	VF	2	00A4
VF PREEMPTION INSTANCE HINT	0x83B4	VF	2	00A6
EXECLIST_STATUS_REGISTER	0x2234	CS	2	00A8
NOOP		CS	20	00AA
NOOP		CS	1	00BE
MI_BATCH_BUFFER_END		CS	1	00BF

Command Ordering Rules

There are several restrictions regarding the ordering of commands issued to the GPE. This subsection describes these restrictions along with some explanation of why they exist. Refer to the various command descriptions for additional information.

The following flowchart illustrates an example ordering of commands which can be used to perform activity within the GPE.



PIPELINE_SELECT

The previously-active pipeline needs to be flushed via the MI_FLUSH command immediately before switching to a different pipeline via use of the PIPELINE_SELECT command. Refer to **Fixed and Shared Function IDs** for details on the PIPELINE_SELECT command.

PIPELINE_SELECT

PIPE_CONTROL

The PIPE_CONTROL command does not require URB fencing/allocation to have been performed, nor does it rely on any other pipeline state. It is intended to be used on both the 3D pipe and the Media pipe. It has special optimizations to support the pipelining capability in the 3D pipe which do not apply to the Media pipe.

Common Pipeline State-Setting Commands

The following commands are used to set state common to both the 3D and Media pipelines. This state is comprised of CS FF unit state, non-pipelined global state (EU, etc.), and Sampler shared-function state.

- STATE_BASE_ADDRESS
- STATE_SIP
- 3DSTATE_SAMPLER_PALETTE_LOAD
- 3DSTATE_CHROMA_KEY

The state variables associated with these commands must be set appropriately prior to initiating activity within a pipeline (i.e., 3DPRIMITIVE or MEDIA_OBJECT).

3D Pipeline-Specific State-Setting Commands

The following commands are used to set state specific to the 3D Pipeline.

- 3DSTATE_PIPELINED_POINTERS
- 3DSTATE_BINDING_TABLE_POINTERS
- 3DSTATE_VERTEX_BUFFERS
- 3DSTATE_VERTEX_ELEMENTS
- 3DSTATE_INDEX_BUFFERS
- 3DSTATE_VF_STATISTICS
- 3DSTATE_DRAWING_RECTANGLE
- 3DSTATE_CONSTANT_COLOR
- 3DSTATE_DEPTH_BUFFER
- 3DSTATE_POLY_STIPPLE_OFFSET
- 3DSTATE_POLY_STIPPLE_PATTERN
- 3DSTATE_LINE_STIPPLE
- 3DSTATE_GLOBAL_DEPTH_OFFSET

The state variables associated with these commands must be set appropriately prior to issuing 3DPRIMITIVE.

Media Pipeline-Specific State-Setting Commands

The following command is used to set state specific to the Media pipeline:

- MEDIA_STATE_POINTERS

The state variables associated with this command must be set appropriately prior to issuing MEDIA_OBJECT.

3DPRIMITIVE

Before issuing a 3DPRIMITIVE command, all state (with the exception of MEDIA_STATE_POINTERS) needs to be valid. Thus the commands used to assigned that state must be issued before issuing 3DPRIMITIVE.

MEDIA_OBJECT

Before issuing a MEDIA_OBJECT command, all state (with the exception of 3D-pipeline-specific state) needs to be valid. Therefore the commands used to set this state need to have been issued at some point prior to the issue of MEDIA_OBJECT.

Resource Streamer

This section contains status registers and controls for the resource streamer.

RS_PREEMPT_STATUS_UDW - RS_PREEMPT_STATUS_UDW

RS_PREEMPT_STATUS - Resource Streamer Preemption Status

MI_RS_CONTEXT

MI_RS_CONTROL

MI_RS_STORE_DATA_IMM

Resource Streamer Sync Commands

Programming Note	
Context:	Resource Streamer Sync Commands.
If resource streamer is enabled in a batch buffer, an MI_RS_STORE_DATA_IMM with Resource Streamer Flush set must be programmed before any Resource Streamer Sync Command.	

Below is a table of commands that cause the resource streamer to stop and wait until the render command streamer restarts the resource streamer. If a command does not end the current batch buffer or disable the resource streamer, then the command streamer will restart the resource streamer before the next command that is used by the resource streamer.

Resource Streamer Sync Commands: Commands that RS Stops
MI_WAIT_FOR_EVENT
MI_RS_CONTROL
MI_BATCH_BUFFER_END
MI_SEMAPHORE_WAIT
MI_SET_CONTEXT
MI_RS_CONTEXT
MI_BATCH_BUFFER_START
MI_CONDITIONAL_BATCH_BUFFER_END

Introduction

The resource streamer is added to offload work from the driver without compromising on GPU optimizations. In order to reduce latency associated with these offloaded operation, H/W adds a Resource Streamer. The Resource Streamer is almost S/W invisible; S/W sees a single command stream, but it may be best for the S/W to be aware that the RS is present, as certain operations might be emphasized. The resource streamer will run ahead of the 3D Command Streamer and process only the certain commands. The Cmd steamer processes these same commands for purposes of buffer full synchronization and buffer consumption.

Glossary

No special terms identified at this time.

Common Abbreviations

CS	**Command Streamer.** Block in charge of streaming commands. The Resource Streamer (RS) is primarily an accelerator for the CS.
FF	**Fixed Function.** Any fixed function hardward.
RS	**The Resource Streamer.** Responsible for reducing command latencies for certain command operations.
URB	**Unified Return Buffer.** The mechanism for returning information from a command.

Theory of Operation

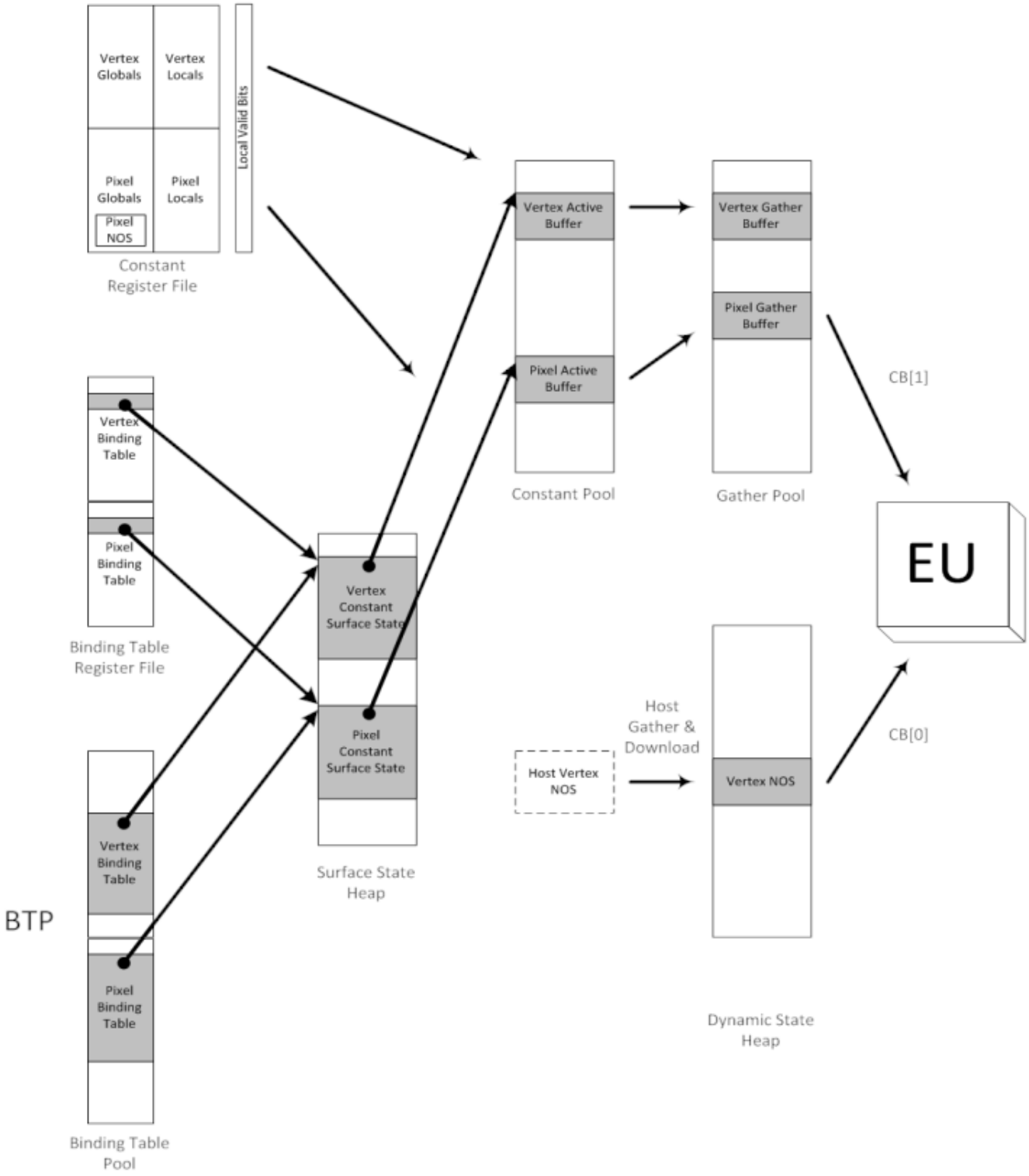
This section briefly describes the operation of the Resource Streamer. Specifically, it calls out reset state, initialization requirements, and major operational tasks of the RS.

Resource Streamer Functions

The Resource Streamer (RS) examines the commands in the ring buffer in an attempt to pre-process certain long latency items for the remainder of the graphics processing. The RS is used for the following operations:

- Batch Processing – The resource streamer reads ahead of command streamer activity to unwind batch buffers.
- Context Save – When the Command Streamer signals that context must be saved, the RS makes certain all previous cycles are completed, saves all context, and signals completion to the command streamer.
- Gather Push Constants – The RS detects GATHER commands (**3DSTATE_GATHER_POOL_ALLOC, 3DSTATE_GATHER_CONSTANT_*****), and prefetchs contents needed for further command processing. The RS gets the base address of the contents by detecting the **3DSTATE_GATHER_POOL_ALLOC** command, and uses other **3DSTATE_GATHER_CONSTANT_*** commands to generate reads for data, and writes out data to memory.
- Constant Buffer Generation – Similar to other constant processes, the RS intercepts the commands for constants to update state and data.
- HW Binding Table Generation/Flush – The RS detects operations in the command stream to update binding table state and memory with bind table contents.

Resource Streamer Activity Diagram



Detailed Resource Streamer Operations

Introduction

This chapter describes the operation of the Resource Streamer in deeper detail. Most of the operations of the Resource Streamer are processed from ring buffer shown in the Ring Buffer Organization Figure in Resource Streamer Operation Descriptions. The RS examines the command stream from the ring buffer to pre-process information required by the 3D Command Streamer (CS). For a large number of the commands, the RS takes no action.

Resource Streamer Operation Descriptions

This section under development.

Batch Processing

When an MI_BATCH_BUFFER_START command is parsed by the render command streamer and the resource streamer enable bit is set, the command stream flags that the resource streamer is enabled. Once it parses a non-sync command then it sends the current address for where the RS must start parsing the batch buffer. The Resource Streamer parses commands until it parses a sync command, which causes the resource streamer to send a message to the command streamer that it has stopped; RS then goes idle. The command streamer stalls at a sync command until the resource streamer specifies it has stopped, and restarts the resource streamer at the next non-sync command. Below is a link to the topic with sync commands.

BSpec > Command Stream Programming > Resource Streamer > Detailed Resource Streamer Operations > Resource Streamer Operation Descriptions > Commands Actions in the RS

Context Save

When the CS indicates that there is a context to be saved or restored, the RS saves its context. The CS provides an address for the RS image and issues a "batch buffer start" (see section *Batch Processing*). The RS consumes this image like any other batch buffer, and stops when it reaches the MI_BATCH_BUFFER_END command.

The context image for the Resource streamer consists of the following components:

1. HW_BINDING_TABLE_IMAGE
2. GATHER_IMAGE
3. CONSTANT_IMAGE
4. MI_BATCH_BUFFER_END

These are discussed in the following subsections.

HW Binding Table Image

While it is not always necessary to save binding table information, “split points” context switches must be saved, so the binding table contents are always saved. These consist of:

- Binding Table Generate Enable
- Binding Table Pool Base Address
- Binding Table Pool Size
- Binding Table Contents

HW Binding Table Image

Description	Dwords Required for Storage
3DSTATE_BINDING_TABLE_POOL_ALLOC	3
3DSTATE_BINDING_TABLE_EDIT_VS	194
3DSTATE_BINDING_TABLE_EDIT_GS	194
3DSTATE_BINDING_TABLE_EDIT_HS	194
3DSTATE_BINDING_TABLE_EDIT_DS	194
3DSTATE_BINDING_TABLE_EDIT_PS	194
3DSTATE_BINDING_TABLE_EDIT_VS	194

Gather Push Constants Image

Since the resource streamer does not support mid-triangle preemption, the resource steamer will have finished producing all the gather buffers by the end of the batch buffer and the cmd streamer would have consumed all the gather buffers. The following things need to be saved.

- Gather pool enable
- Gather pool base address
- Gather pool size

Therefore a 3DSTATE_GATHER_POOL_ALLOC command needs to be saved.

Gather Push Constants Image

Description	Dwords Required for Storage
3DSTATE_GATHER_POOL_ALLOC	4

Push Constant Image

We assume that the end of the batch buffer can come between any set of cmds. Therefore the following things will be saved:

- Dx9 Constant enable
- Dx9 Constant pool base address
- Dx9 Constant pool size
- Dx9 local registers (F,I,B)
- Dx9 Local Valid
- Dx9 global registers (F,I,B)

Therefore a 3DSTATE_CONSTANT_BUFFER_POOL_ALLOC command will saved. In addition, since the F register is 256 entries and only a maximum of 63 entries can be contained in a single 3DSTATE_DX9CONSTANTF_* command, 5 CONSTANTF cmds will be saved for global and 5 for local registers register per FF (VS,PS). There will be 1 3DSTATE_CONSTANTI_* will be save for global and 1 for local register per FF. There will be 1 3DSTATE_CONSTANTB_* will be save for global and 1 for local register per FF.

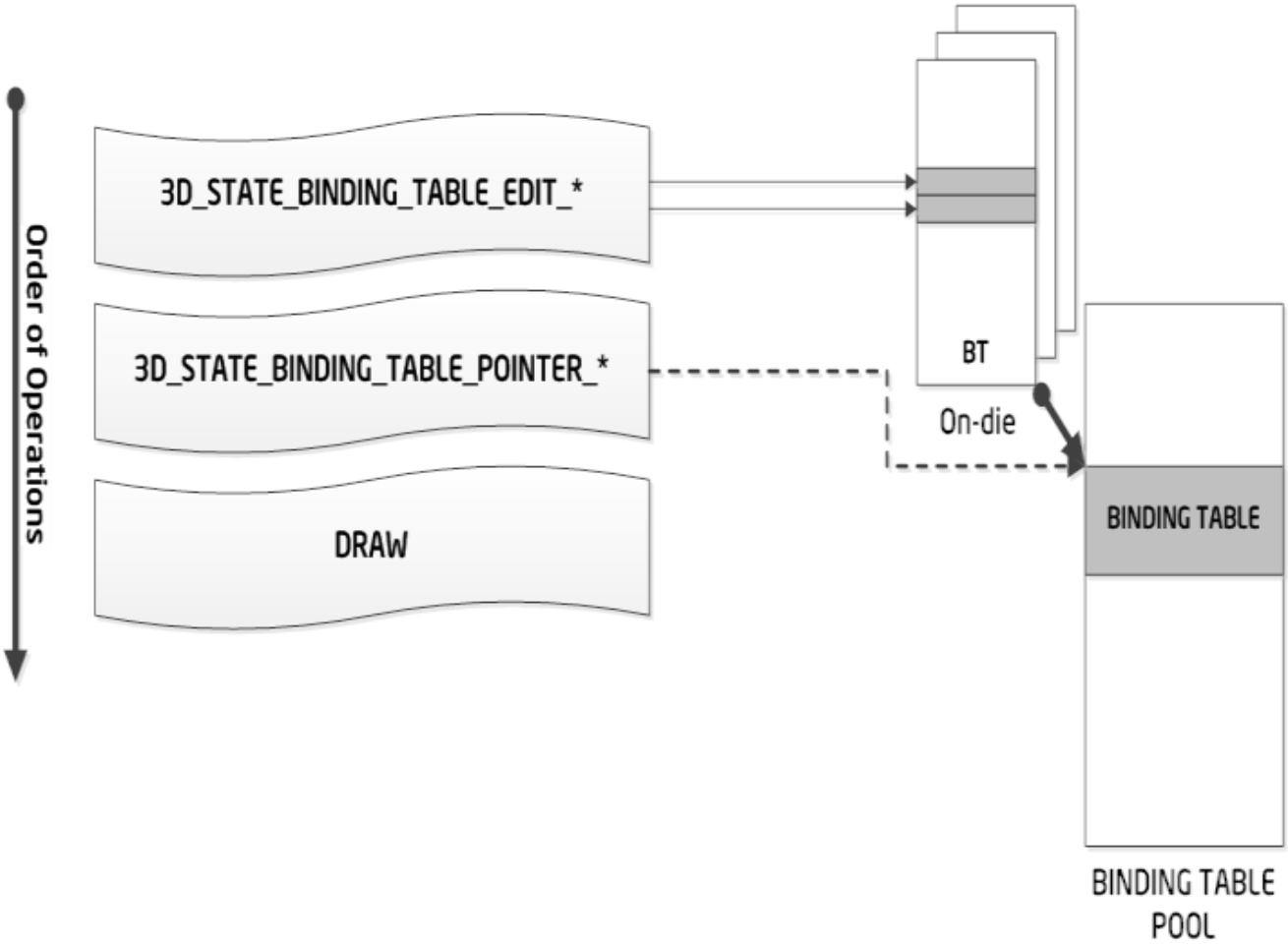
Gather Push Constants Image

Description	Dwords Required for Storage
3DSTATE_CONSTANT_BUFFER_POOL_ALLOC	4
3DSTATE_CONSTANTF_VS	1026
3DSTATE_CONSTANTI_VS	130
3DSTATE_CONSTANTB_VS	18
3DSTATE_CONSTANTF_VS	1026
3DSTATE_CONSTANTI_VS	130
3DSTATE_CONSTANTB_VS	18
3DSTATE_LOCAL_VALID_VS	10
3DSTATE_CONSTANTF_PS	1026
3DSTATE_CONSTANTI_PS	130
3DSTATE_CONSTANTB_PS	18
3DSTATE_CONSTANTF_PS	1026
3DSTATE_CONSTANTI_PS	130
3DSTATE_CONSTANTB_PS	18
3DSTATE_LOCAL_VALID_PS	10

HW Binding Table Generation

The RS generates binding tables in hardware to offload this from the driver. There is an on-die set of binding tables for each fixed-function unit (VS, GS, HS, DS, and PS). There is a set of commands generated by the driver to update each of these tables (3D_STATE_BINDING_TABLE_POINTER_*). When the RS encounters any of these commands, it writes the corresponding binding table out to the binding table pool. When the CS encounters these commands, it sends the binding table points down as pipelined state.

HW Binding Table Generation



The following table describes the different types of usages with binding table generation.

RS Active *	BT Pool Enabled	Mode
0	0	SW Generate BT in Surface State Heap
0	1	Illegal (Undefined)
1	0	Illegal **
1	1	HW Generate BT

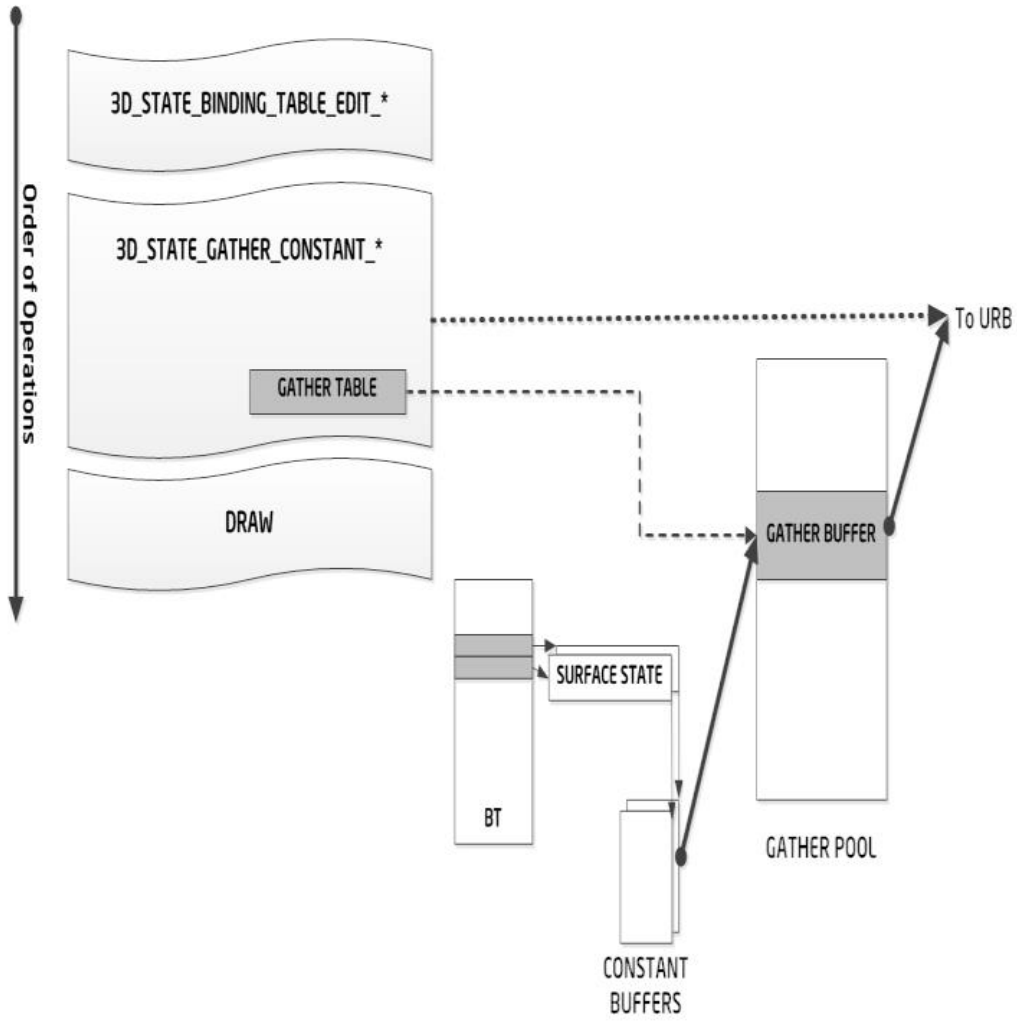
Gather Push Constants

Applications can provide up to 16 constant buffers. The compiler does some optimizations of constant usage and determines which constants should be packed in which constant register for optimal shader performance. While this gathering and packing of constant elements into push constants optimizes the shaders, it causes the driver additional work at draw call time, since the driver must gather and pack the constants at draw time.

The RS offloads the gathering process for the driver by interpreting the 3D_STATE_GATHER_CONSTANT_* for each of the fixed functions (VS, GS, DS, HS, PS). The compiler generates at gather table which instructs which elements of the buffers should be packed into the gather buffer. The gather table indexes the binding table to get a surface state which in turn points to the constant buffer. Once the gather buffer has been filled, the CS will execute the 3D_STATE_GATHER_CONSTANT_* to load the push constant into the URB.

Note: The gather push constants can ONLY BE USED if the HW generated binding tables are also used.

Gather Push Constants Generation

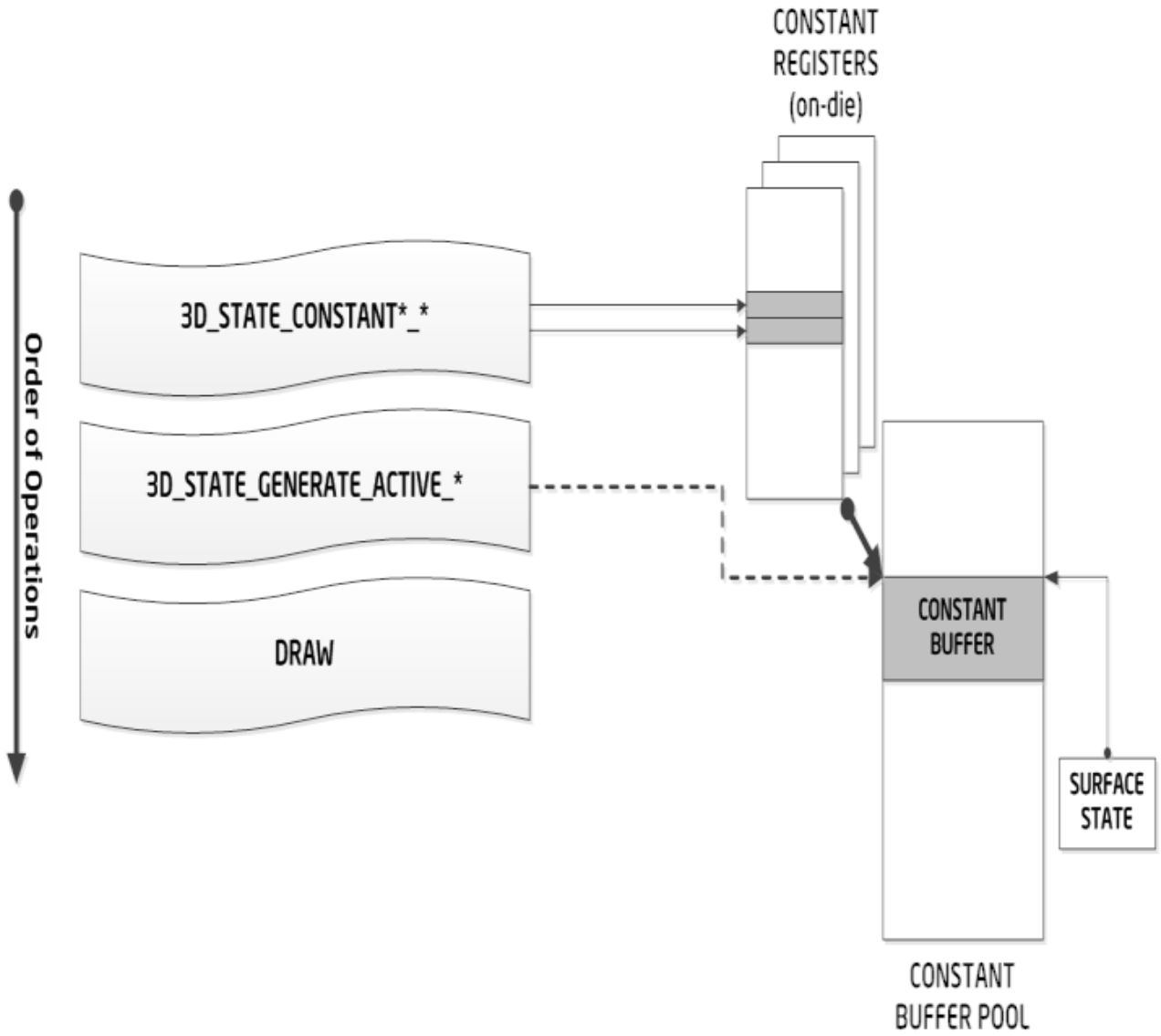


Constant Buffer Generation (not DX9)

The constant model used is a set of registers that the application can incrementally update. The hardware requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3D_STATE_CONSTANT*_* commands are used. The constant registers can be either floating, integer, or Boolean (signified by the commands CONSTANTF, CONSTANTI, or CONSTANTB, respectively). The option determines the fixed function for the constants (VS, GS, DS, HS, or PS).

When all edits to the constant registers have been completed, the 3D_STATE_GENERATE_ACTIVE_* command is used to write out a constant buffer to the Constant Buffer Pool. These buffers are fixed at 8Kbytes. Software is required to provide a surface state object that points to the constant buffer created.

Constant Buffer Generation



Commands Actions in the RS

The tables below show all 3D commands processed by the RS. In the following tables, "STOP" indicates that the RS waits for all engines to complete operations AND invalidates all command data currently in the command FIFO. "BLOCK" indicates that the RS waits for all engines to complete operation, stops further command parsing, but retains data in the command FIFO.

MI Commands Processing in the RS

Opcode (28:23)	Command	RS Handing (No Perf)	RS Handling (Perf)
03h	MI_WAIT_FOR_EVENT	STOP	BLOCK
05h	MI_ARB_CHECK	STOP	STOP
06h	MI_RS_CONTROL	STOP	STOP
0Ah	MI_BATCH_BUFFER_END	STOP	STOP
16h	MI_SEMAPHORE_MBOX	STOP	BLOCK
18h	MI_SET_CONTEXT	STOP	STOP
1Ah	MI_RS_CONTEXT	STOP	STOP
31h	MI_BATCH_BUFFER_START	STOP	STOP
36h	MI_CONDITIONAL_BATCH_BUFFER_END	STOP	STOP

Other Commands Processed in the RS

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
0h	1h	01h	STATE_BASE_ADDRESS	RS LATCH	RS LATCH	RSunit updates the state base address if parsed
1h	1h	04h	PIPELINE_SELECT	STOP	STOP	Stop only if 3D is not selected
3h	0h	03h	Reserved			
3h	0h	04h	3DSTATE_CLEAR_PARAMS			Refer to 3D Pipeline
3h	0h	05h	3DSTATE_DEPTH_BUFFER			Refer to 3D Pipeline
3h	0h	06h	Reserved			
3h	0h	06h	3DSTATE_STENCIL_BUFFER			Refer to 3D Pipeline
3h	0h	07h	Reserved			
3h	0h	07h	3DSTATE_HIER_DEPTH_BUFFER			Refer to 3D

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
						Pipeline
3h	0h	08h	3DSTATE_VERTEX_BUFFERS			Refer to Vertex Fetch
3h	0h	09h	3DSTATE_VERTEX_ELEMENTS			Refer to Vertex Fetch
3h	0h	0Ah	3DSTATE_INDEX_BUFFER			Refer to Vertex Fetch
3h	0h	0Bh	3DSTATE_VF_STATISTICS			Refer to Vertex Fetch
3h	0h	0Ch	Reserved			
3h	0h	0Dh	3DSTATE_VIEWPORT_STATE_POINTERS			Refer to 3D Pipeline
3h	0h	10h	3DSTATE_VS			Refer to Vertex Shader
3h	0h	11h	3DSTATE_GS			Refer to Geometry Shader
3h	0h	12h	3DSTATE_CLIP			Refer to Clipper
3h	0h	13h	3DSTATE_SF			Refer to Strips and Fans
3h	0h	14h	3DSTATE_WM			Refer to Windower
3h	0h	15h	3DSTATE_CONSTANT_VS			Refer to Vertex Shader
3h	0h	16h	3DSTATE_CONSTANT_GS			Refer to Geometry Shader
3h	0h	17h	3DSTATE_CONSTANT_PS			Refer to Windower
3h	0h	18h	3DSTATE_SAMPLE_MASK			Refer to Windower
3h	0h	19h	3DSTATE_CONSTANT_HS			Refer to Hull Shader
3h	0h	1Ah	3DSTATE_CONSTANT_DS			Refer to Domain Shader
3h	0h	1Bh	3DSTATE_HS			Refer to Hull Shader

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	1Ch	3DSTATE_TE			Refer to Tesselator
3h	0h	1Dh	3DSTATE_DS			Refer to Domain Shader
3h	0h	1Eh	3DSTATE_STREAMOUT			Refer to HW Streamout
3h	0h	1Fh	3DSTATE_SBE			Refer to Setup
3h	0h	20h	3DSTATE_PS			Refer to Pixel Shader
3h	0h	21h	Reserved			
3h	0h	22h	3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP			Refer to Strips & Fans
3h	0h	23h	3DSTATE_VIEWPORT_STATE_POINTERS_CC			Refer to Windower
3h	0h	24h	3DSTATE_BLEND_STATE_POINTERS			Refer to Pixel Shader
3h	0h	25h	3DSTATE_DEPTH_STENCIL_STATE_POINTERS			Refer to Pixel Shader
3h	0h	26h	3DSTATE_BINDING_TABLE_POINTERS_VS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	27h	3DSTATE_BINDING_TABLE_POINTERS_HS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	28h	3DSTATE_BINDING_TABLE_POINTERS_DS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	29h	3DSTATE_BINDING_TABLE_POINTERS_GS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	2Ah	3DSTATE_BINDING_TABLE_POINTERS_PS	Generate BT if HW BT enabled	Generate BT if HW BT enabled	
3h	0h	2Fh	Reserved			
3h	0h	30h	3DSTATE_URB_VS/td>	Execute	Execute	
3h	0h	31h	3DSTATE_URB_HS	Execute	Execute	
3h	0h	32h	3DSTATE_URB_DS	Execute	Execute	

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	33h	3DSTATE_URB_GS	Execute	Execute	
3h	0h	34h	3DSTATE_GATHER_VS	Execute	Execute	
3h	0h	35h	3DSTATE_GATHER_GS	Execute	Execute	
3h	0h	36h	3DSTATE_GATHER_HS	Execute	Execute	
3h	0h	37h	3DSTATE_GATHER_DS	Execute	Execute	
3h	0h	38h	3DSTATE_GATHER_PS	Execute	Execute	
3h	0h	39h	3DSTATE_CONSTANTF_VS	Execute	Execute	
3h	0h	3Ah	3DSTATE_CONSTANTF_PS	Execute	Execute	
3h	0h	3Bh	3DSTATE_CONSTANTI_VS	Execute	Execute	
3h	0h	3Ch	3DSTATE_CONSTANTI_PS	Execute	Execute	
3h	0h	3Dh	3DSTATE_CONSTANTB_VS	Execute	Execute	
3h	0h	3Eh	3DSTATE_CONSTANTB_PS	Execute	Execute	
3h	0h	3Fh	3DSTATE_LOCAL_VALID_VS	Execute	Execute	
3h	0h	40h	3DSTATE_LOCAL_VALID_PS	Execute	Execute	
3h	0h	41h	3DSTATE_GENERATE_ACTIVE_VS	Execute	Execute	
3h	0h	42h	3DSTATE_GENERATE_ACTIVE_PS	Execute	Execute	
3h	0h	43h	3DSTATE_BINDING_TABLE_EDIT_VS			Refer to Vertex Shader
3h	0h	44h	3DSTATE_BINDING_TABLE_EDIT_GS			Refer to Vertex Shader
3h	0h	45h	3DSTATE_BINDING_TABLE_EDIT_HS			Refer to Vertex Shader
3h	0h	46h	3DSTATE_BINDING_TABLE_EDIT_DS			Refer to Vertex Shader
3h	0h	47h	3DSTATE_BINDING_TABLE_EDIT_PS			Refer to Vertex Shader
3h	0h	48h	3DSTATE_VF_HASHING			
3h	0h	49h	3DSTATE_VF_INSTANCING			
3h	0h	4Ah	3DSTATE_VF_SGVS			
3h	0h	4Bh	3DSTATE_VF_TOPOLOGY			
3h	0h	4Ch	3DSTATE_WM_CHROMA_KEY			
3h	0h	4Dh	3DSTATE_PS_BLEND			
3h	0h	4Eh	3DSTATE_WM_DEPTH_STENCIL			
3h	0h	4Fh	3DSTATE_PS_EXTRA			
3h	0h	50h	3DSTATE_RASTER			
3h	0h	51h	3DSTATE_SBE_SWIZ			

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	0h	52h	3DSTATE_WM_HZ_OP			
3h	0h	53h	3DSTATE_INT (internally generated state)			
3h	0h	57h	3DSTATE_DX9_CONSTANTF_HS			
3h	0h	58h	3DSTATE_DX9_CONSTANTI_HS			
3h	0h	59h	3DSTATE_DX9_CONSTANTB_HS			
3h	0h	5ah	3DSTATE_DX9_LOCAL_VALID_HS			
3h	0h	5bh	3DSTATE_DX9_GENERATE_ACTIVE_HS			
3h	0h	5ch	3DSTATE_DX9_CONSTANTF_DS			
3h	0h	5dh	3DSTATE_DX9_CONSTANTI_DS			
3h	0h	5eh	3DSTATE_DX9_CONSTANTB_DS			
3h	0h	5fh	3DSTATE_DX9_LOCAL_VALID_DS			
3h	0h	60h	3DSTATE_DX9_GENERATE_ACTIVE_DS			
3h	0h	61h	3DSTATE_DX9_CONSTANTF_GS			
3h	0h	62h	3DSTATE_DX9_CONSTANTI_GS			
3h	0h	63h	3DSTATE_DX9_CONSTANTB_GS			
3h	0h	64h	3DSTATE_DX9_LOCAL_VALID_GS			
3h	0h	65h	3DSTATE_DX9_GENERATE_ACTIVE_GS			
3h	0h	67h-FFh	Reserved			
3h	1h	00h	3DSTATE_DRAWING_RECTANGLE			
3h	1h	02h	3DSTATE_SAMPLER_PALETTE_LOAD0			
3h	1h	03h	Reserved			
3h	1h	04h	3DSTATE_CHROMA_KEY			
3h	1h	05h	Reserved			
3h	1h	06h	3DSTATE_POLY_STIPPLE_OFFSET			
3h	1h	07h	3DSTATE_POLY_STIPPLE_PATTERN			
3h	1h	08h	3DSTATE_LINE_STIPPLE			
3h	1h	0Ah	3DSTATE_AA_LINE_PARAMS			
3h	1h	0Bh	3DSTATE_GS_SVB_INDEX			
3h	1h	0Ch	3DSTATE_SAMPLER_PALETTE_LOAD1			
3h	1h	0Dh	3DSTATE_MULTISAMPLE			
3h	1h	0Eh	3DSTATE_STENCIL_BUFFER			
3h	1h	0Fh	3DSTATE_HIER_DEPTH_BUFFER			
3h	1h	10h	3DSTATE_CLEAR_PARAMS			
3h	1h	11h	3DSTATE_MONOFILTER_SIZE			

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	RS Handling (No Perf)	RS Handling (Perf)	Notes
3h	1h	12h	3DSTATE_PUSH_CONSTANT_ALLOC_VS			
3h	1h	13h	3DSTATE_PUSH_CONSTANT_ALLOC_HS			
3h	1h	14h	3DSTATE_PUSH_CONSTANT_ALLOC_DS			
3h	1h	15h	3DSTATE_PUSH_CONSTANT_ALLOC_GS			
3h	1h	16h	3DSTATE_PUSH_CONSTANT_ALLOC_PS			
3h	1h	17h	3DSTATE_SO_DECL_LIST			
3h	1h	18h	3DSTATE_SO_BUFFER			
3h	1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC			
3h	1h	1Ah	3DSTATE_GATHER_POOL_ALLOC			
3h	1h	1Bh	3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC			
3h	1h	1Ch	3DSTATE_SAMPLE_PATTERN			
3h	1h	1Dh-FFh	Reserved			
3h	1h	19h	3DSTATE_BINDING_TABLE_POOL_ALLOC	Execute	Execute	
3h	1h	1Ah	3DSTATE_GATHER_POOL_ALLOC	Execute	Execute	
3h	1h	1Bh	3DSTATE_CONSTANT_BUFFER_POOL_ALLOC	Execute	Execute	
3h	1h	1Ch	3DSTATE_SAMPLE_PATTERN			
3h	1h	1Dh-FFh	Reserved			
3h	2h	00h	PIPE_CONTROL			
3h	2h	01h-FFh	Reserved			
3h	3h	00h	3DPRIMITIVE	Sync	Sync	3DPRIMITIVE command is unique in that it tells the engines to send fence cycles, but does not stop RSunit (not a sync point)

Resource Streamer Programming Guidelines

This section describes RS activities and assumptions that are required for programming.

RS Interactions with the 3D Command Streamer

Because the Resource Streamer is processing ahead of the Command Streamer, many of the commands interpreted by the RS are a signal to stop further processing. In these cases, the RS completes pending activity, and waits for an indication from the Command Streamer to start again.

The specific cases that the CS commands the RS to continue are:

- Batch Buffer command parsing
- Context save

RS Interactions with Memory Requests

The RS is responsible for the generation of a number of memory requests. These are:

- Make batch buffer read requests (when address is supplied from the CS).
- Make push constant gather read requests from the state base offset.
- Make push constant gather write of packed data to the gather pool.
- Fetch the gather buffer surface base address.
- Write out the binding table pointer (BTP).
- Save BTP, constant buffer, and gather constant context data to an offset into the context image.
- Write out constant data.

As is the case in all memory accesses, read requests from the RS can be freely reordered, and may be returned in any order by the hardware. The RS consumes the cycles and presents the “software” order transparently.

When accessing the same address, a write operation followed by the read returns the written data. Writes to non-overlapping addresses may be freely reordered as well. Fencing is used to make certain all writes up to the fence have completed.

Fundamental Programming and Operational Assumptions

The following assumptions are made in the RS, and these are useful limitations to the programming:

- The CS can never send a request to a busy RS. The RS will have foreseen the situation, and stopped its operations before the CS action.
- Surface base address is never changed while in a batch buffer.
- Push constant data is 128-bit aligned.
- The GATHER command should have Constant Buffer valid bits set for any indices used in the command.

Non-Operational Activities

There are no specific events or performance counters for the resource streamer (RS).

Hardware Binding Tables

The driver spends a considerable amount of time managing the binding tables. A new command is added, 3DSTATE_BINDING_TABLE_EDIT_*, to offload the binding table generation from the driver. There is an on-die set of binding tables for each FF (VS, GS, HS, DS, PS). The 3DSTATE_BINDING_TABLE_EDIT_* commands are used by the driver to update these tables. The 3DSTATE_BINDING_TABLE_POINTER_* commands are added. When the resource streamer encounters a 3DSTATE_BINDING_TABLE_POINTER_* command, it writes the binding table out to the binding table pool. When the command streamer encounters a 3DSTATE_BINDING_TABLE_POINTER_* command, it sends the binding table pointer down as pipelined state.

Hardware Binding Tables are only supported for 3D workloads. The resource streamer must be enabled only for 3D workloads. The resource streamer must be disabled for Media and GPGPU workloads. A batch buffer containing both 3D and GPGPU workloads must take care of disabling and enabling the Resource Streamer appropriately while changing the PIPELINE_SELECT mode from 3D to GPGPU and vice versa. The resource streamer must be disabled using MI_RS_CONTROL command and Hardware Binding Tables must be disabled by programming 3DSTATE_BINDING_TABLE_POOL_ALLOC with "Binding Table Pool Enable" set to disable (i.e value '0'). The following example shows disabling and enabling of the resource streamer in a batch buffer for 3D and GPGPU workloads:

```
MI_BATCH_BUFFER_START (Resource Streamer Enabled)
PIPELINE_SELECT (3D)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled)
3D WORKLOAD MI_RS_CONTROL (Disable Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Disabled)
PIPELINE_SELECT (GPGPU)
GPGPU Workload
PIPELINE_SELECT (3D)
MI_RS_CONTROL (Enable Resource Streamer)
3DSTATE_BINDING_TABLE_POOL_ALLOC (Binding Table Pool Enabled)
3D WORKLOAD
MI_BATCH_BUFFER_END
```

3DSTATE_BINDING_TABLE_POOL_ALLOC [CHV, BSW]

Programming Note
The binding table generator feature has a simple all or nothing model. If HW generated binding tables are enabled, the driver must enable the pool and use 3D_HW_BINDING_TABLE_POINTER_* commands.
When switching between HW and SW binding table generation, SW must issue a state cache invalidate.
A maximum of 16,383 Binding Tables are allowed in any batch buffer.
If the Binding Table Pool Enable is cleared while the Resource Streamer is enabled within a batch buffer, then the on chip storage for the binding table will not be context save and restored. To save the Binding Table Pool, before disabling the Pool enable, disable the resource streamer thru the MI_RS_CONTROL command. And then, before reenabling the Binding Table Pool, reenable the resource streamer thru the MI_RS_CONTROL command.

The variable length commands are 3DSTATE_BINDING_TABLE_EDIT_HS, 3DSTATE_BINDING_TABLE_EDIT_DS, and 3DSTATE_BINDING_TABLE_EDIT_PS.

3DSTATE_BINDING_TABLE_POOL_ALLOC

3DSTATE_BINDING_TABLE_EDIT_VS

3DSTATE_BINDING_TABLE_EDIT_HS

3DSTATE_BINDING_TABLE_EDIT_DS

3DSTATE_BINDING_TABLE_EDIT_GS

3DSTATE_BINDING_TABLE_EDIT_PS

Gather Constants

Gather commands support fetching from 16 different constant buffers or one constant buffer of 8KB size. The compiler does some optimizations of constant usage and determines which elements of which constants should be packed in which push constant register for optimum shader performance. While this gathering and packing of constant elements into push constant registers optimizes the shader, it causes the driver added work at draw call time, because the driver must do the gather and packing at draw time. New commands (3D_STATE_GATHER_CONSTANT_* and 3DSTATE_GATHER_POOL_ALLOC) were added to offload the gather and packing functions from the driver. The base address for the push constant buffer and the enabling of the feature is programmed through the 3DSTATE_GATHER_POOL_ALLOC. There are 5 FF which support push constants (VS, GS, DS, HS, PS) and they all have corresponding gather commands. The compiler generates a gather table that specifies what elements of what buffers are packed into the gather buffer. The gather table indexes the BT to get the surface state which points to the constant buffer. The resource streamer gathers constants by reading the constant buffer, packs the data and then writes the buffer out to a push constant buffer based on the base address and the offset in the 3DSTATE_GATHER_CONSTANT_* command.

3DSTATE_GATHER_POOL_ALLOC

3DSTATE_GATHER_CONSTANT_VS

3DSTATE_GATHER_CONSTANT_HS

3DSTATE_GATHER_CONSTANT_DS

3DSTATE_GATHER_CONSTANT_GS

3DSTATE_GATHER_CONSTANT_PS

Dx9 Constant Buffer Generation

The Dx9 constant model is a set of register that the App can incrementally update. The HW requires a constant buffer which lives until the last shader using that buffer retires. To offload the driver the 3DSTATE_DX9_CONSTANT*_* cmds are added. These commands allow the on-die constant register to be maintained. When all the edits to the constant register have been completed, the 3DSTATE_DX9_GENERATE_ACTIVE_* cmd is used to write out a constant buffer to the Dx9 Constant buffer pool. The Dx9 constant buffers are fixed 8KB in size, with a large portion of the second 4KB unused.

Programming Note	
Context:	Dx9 Constant Buffer generation
For buffers, which have no inherent "height," padding requirements are different. A buffer must be padded to the next multiple of 256 array elements, with an additional 16 bytes added beyond that to account for the L1 cache line.	

Programming Note	
Context:	Dx9 Constant Buffer generation.
• The Dx9 constant buffer feature has a simple all or nothing model. • A maximum of 16,383 Binding Tables are allowed in any batch buffer. • The Dx9 constants can only be enabled if the binding table generator is also enabled.	

3DSTATE_DX9_CONSTANT_BUFFER_POOL_ALLOC

Vertex Shader Constant

This section contains various commands for the vertex shader constant.

3DSTATE_DX9_CONSTANTF_VS

3DSTATE_DX9_CONSTANTI_VS

3DSTATE_DX9_CONSTANTB_VS

3DSTATE_DX9_LOCAL_VALID_VS

DX9 Constant Buffer Map [CHV, BSW]

Offset			Cache Line
0x0000	256 4-component Float Constants		0
0x0fff			63
0x1000	32 4-component integer constants		64
0x107f			
0x1080	16 1-component boolean constants		72
0x108f			
0x1090	unused		
0x1fff			

3DSTATE_DX9_GENERATE_ACTIVE_VS

Pixel Shader Constant

This section contains various commands for the pixel shader constant.

3DSTATE_DX9_CONSTANTF_PS

3DSTATE_DX9_CONSTANTI_PS

3DSTATE_DX9_CONSTANTB_PS

3DSTATE_DX9_LOCAL_VALID_PS

DX9 Constant Buffer Map [CHV, BSW]

Offset			Cache Line
0x0000	256 4-component Float Constants		0
0x0fff			63
0x1000	32 4-component integer constants		64
0x107f			
0x1080	16 1-component boolean constants		72
0x108f			
0x1090	unused		
0x1fff			

3DSTATE_DX9_GENERATE_ACTIVE_PS

Shared Functions

3D Sampler

The 3D Sampling Engine provides the capability of advanced sampling and filtering of surfaces in memory.

The sampling engine function is responsible for providing filtered texture values to the Gen Core in response to sampling engine messages. The sampling engine uses SAMPLER_STATE to control filtering modes, address control modes, and other features of the sampling engine. A pointer to the sampler state is delivered with each message, and an index selects one of 16 states pointed to by the pointer. Some messages do not require SAMPLER_STATE. In addition, the sampling engine uses SURFACE_STATE to define the attributes of the surface being sampled. This includes the location, size, and format of the surface as well as other attributes.

Although data is commonly used for “texturing” of 3D surfaces, the data can be used for any purpose once returned to the execution core. The 3D Sampler can be used to assist the media sampler in specific operations such as video scaling.

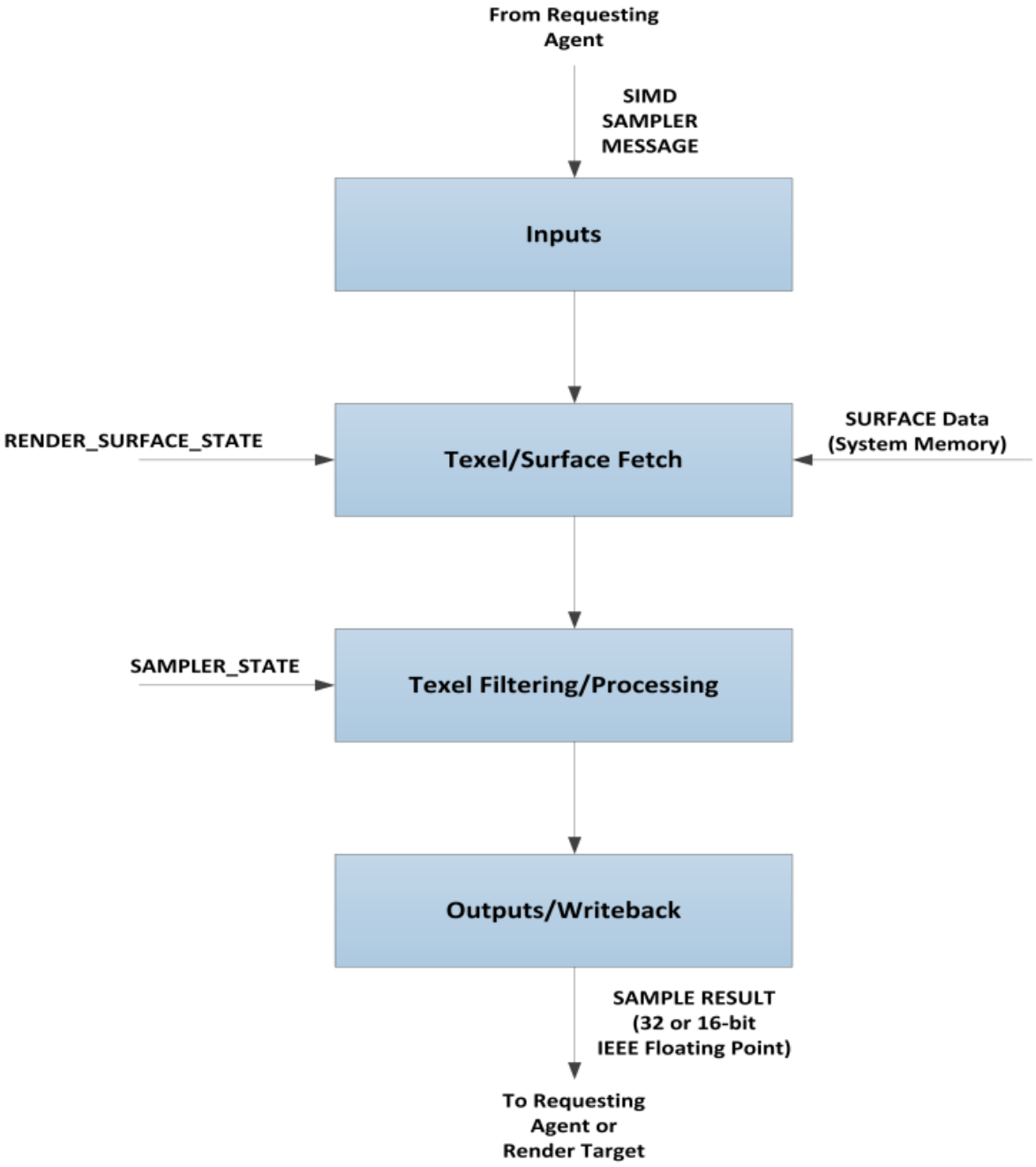
The following table summarizes the various subfunctions provided by the Sampling Engine. After the appropriate subfunctions are complete, the 4-component (reduced to fewer components in some cases) filtered texture value is provided to the Gen Core to complete the *sample* instruction.

Subfunction	Description
Texture Coordinate Processing	Any required operations are performed on the incoming pixel's interpolated internal texture coordinates. These operations may include cube map intersection.
Texel Address Generation	The Sampling Engine determines the required set of texel samples (specific texel values from specific texture maps), as defined by the texture map parameters and filtering modes. This includes coordinate wrap/clamp/mirror control, mipmap LOD computation and sample and/or miplevel weighting factors to be used in the subsequent filtering operations.
Texel Fetch	The required texel samples are read from the texture map. This step may require decompression of texel data. The texel sample data is converted to an internal format.
Texture Palette Lookup	For streams which have “paletted” texture surface formats, this function uses the “index” values read from the texture map to look up texel color data from the texture palette.
Shadow Pre-Filter Compare	For shadow mapping, the texel samples are first compared to the 3rd (R) component of the pixel’s texture coordinate. The boolean results are used in the texture filter.
Texel Filtering	Texel samples are combined using the filter weight coefficients computed in the Texture Address Generation function. This “combination” ranges from simply passing through a “nearest” sample to blending the results of anisotropic filters performed on two mipmap levels. The output of this function is a single 4-component texel value.
Texel Color Gamma Linearization	Performs optional gamma decorrection on texel RGB (not A) values.
8x8 Video Scaler	Performs scaling using an 8x8 filter.

Sampling Engine

3D Sampler Theory of Operation

The 3D sampler (sometimes referred to as texture sampler) is a self-contained functional block within the Graphics Core which receives messages from other agents in the Graphics Core, fetches data from external memory sources typically referred to as "surfaces", performs operations on the data and returns the results in standard formats to the requester (or directly to a Render Target is requested). One of the most common applications of the 3D sampler is to return a filtered/blended pixel from a location in a texture map.



Sampler Inputs Messages

Input requests to the 3D Sampler are in the form of messages (see Messages sub-section for a description of message types and formats). A pixel shader kernal executing on the Graphics Core is an example of an agent which is capable of sending sample messages to the 3D Sampler.

In its most basic form, the sampler receives coordinates to a location within a field of data (often a texture map or depth map) and returns a value which represents the pixel color or depth which may be filtered/blended as defined by associated surface and sampler state objects. Sampler can also work on un-typed data structures called buffers.

Messages are sent in SIMD (Single Instruction Multiple Data) format where there are 8, 16, 32 or 64 coordinate tuples to be processed (i.e. SIMD8, SIMD16 etc.) in the same manner. Some message types are restricted to SIMD8 and SIMD16 varieties and other are restricted to SIMD32 or SIMD64. See the section on Texture Coordinate Processing more details on texture coordinate requirements.

SIMD8 and SIMD16 messages are further organized into groups of 4 sets of coordinates which generally form a 2x2 “subspan” of texel locations. The spatial locality of the texel locations within a sub-span improves the performance of the sampler and allows the processing of the 4 texel locations in parallel. A SIMD8 message contains two subspans and a SIMD16 contains 4 subspans.

Sampler Data Fetches

The 3D sampler will automatically fetch required data from surfaces in system memory as needed to perform each sample operation. Fetched data may be stored in an internal cache to reduce latency for subsequent fetch operations.

The sampler calculates the address into a surface and uses RENDER_SURFACE_STATE state objects to determine the location within system memory and the format of the surface being fetched. Sampler can also receive or calculate the LOD (Level of Detail) of a surface if the surface supports multiple Mips and will fetch from the correct Mip in this case. See Texture Address Calculation sub-section for more detail on addresses and LOD calculation.

The sampler will also automatically decompress any supported compression format once data has been fetched. See the subsection Surface State for a list of supported surface formats, including compressed formats. Likewise, the sampler can linearize (inverse Gamma) sRGB formats prior to filtering.

Sampler Filtering and Processing

The sampler is capable of performing all basic filtering operations (point, bilinear, trilinear, anisotropic, cube etc.) based on the SAMPLER_STATE state object associated with the sample operation being requested.

In most cases, data returned is in the form of 32-bit or 16-bit IEEE floating-point per channel to ensure maximum precision. See Writeback Message section for a description of the format of returned data. Output Data is only returned to the requesting agent or written to a designated Render Target (RT). Sample results are never cached within the sampler or written to system memory.

Texture Coordinate Processing

The Texture Coordinate Processing function of the Sampling Engine performs any operations on the texture coordinates that are required before physical addresses of texel samples can be generated.

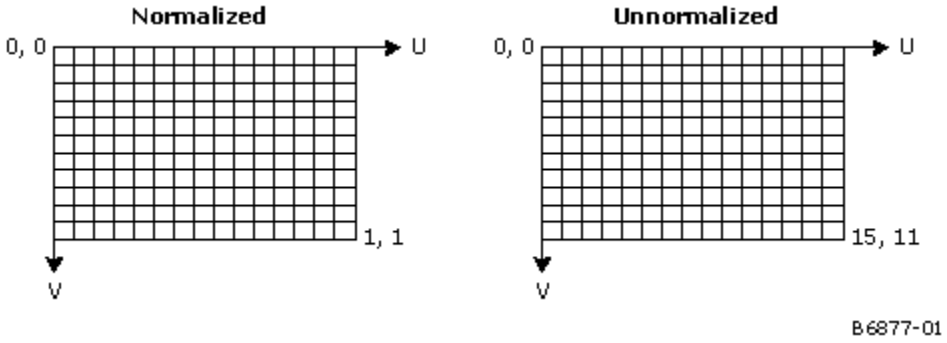
Texture Coordinate Normalization

A texture coordinate may have *normalized* or *unnormalized* values. In this function, unnormalized coordinates are normalized.

Normalized coordinates are specified in units relative to the map dimensions, where the origin is located at the upper/left edge of the upper left texel, and the value 1.0 coincides with the lower/right edge of the lower right texel . 3D rendering typically utilizes normalized coordinates.

Unnormalized coordinates are in units of texels and have not been divided (normalized) by the associated map's height or width. Here the origin is the located at the upper/left edge of the upper left texel of the base texture map.

Normalized vs. Unnormalized Texture Coordinates



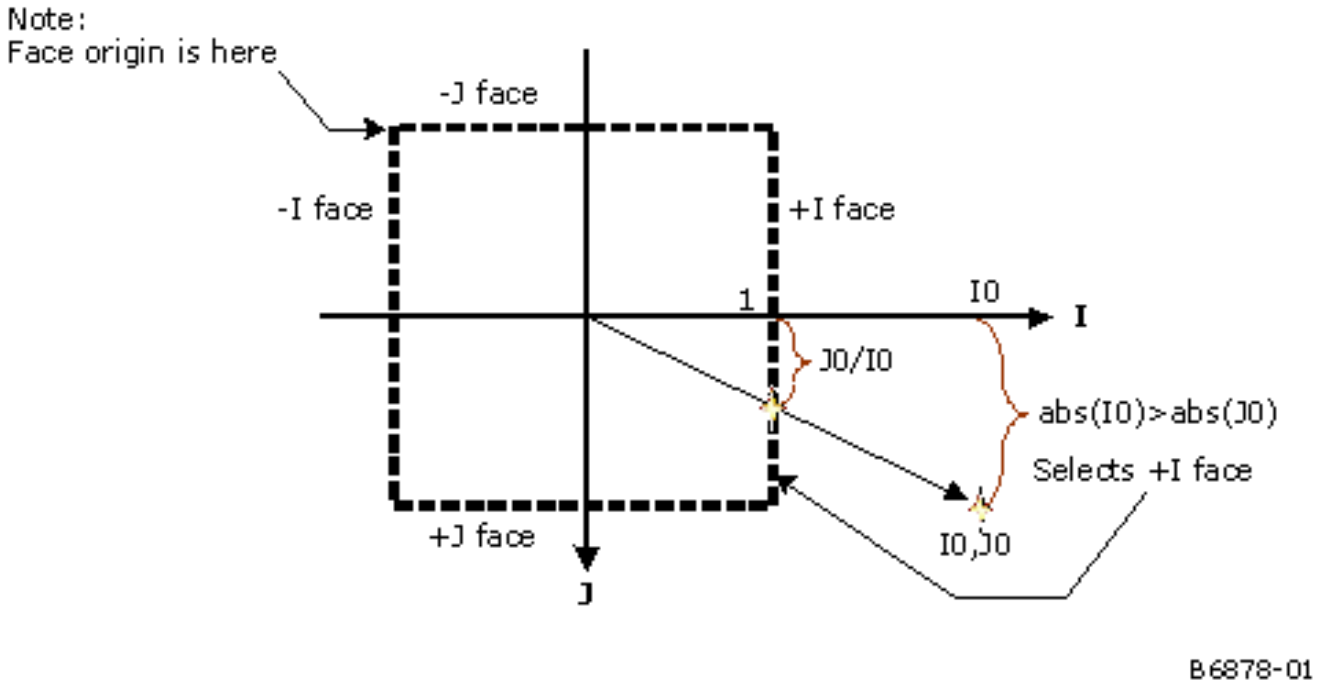
Texture Coordinate Computation

Cartesian (2D) and homogeneous (projected) texture coordinate values are projected from (interpolated) screen space back into texture coordinate space by dividing the pixel's S and T components by the Q component. This operation is done prior to sending sample operations to the 3D sampler.

Vector (cube map) texture coordinates are generated by first determining which of the 6 cube map faces (+X, +Y, +Z, -X, -Y, -Z) the vector intersects. The vector component (X, Y or Z) with the largest absolute value determines the proper (major) axis, and then the sign of that component is used to select between the two faces associated with that axis. The coordinates along the two minor axes are then divided by the coordinate of the major axis, and scaled and translated, to obtain the 2D texture coordinate ([0,1]) within the chosen face. Note that the coordinates delivered to the sampling engine must already have been divided by the component with the largest absolute value.

An illustration of this cube map coordinate computation, simplified to only two dimensions, is provided below:

Cube Map Coordinate Computation Example



Texel Address Generation

To better understand texture mapping, consider the mapping of each object (screen-space) pixel onto the textures images. In texture space, the pixel becomes some arbitrarily sized and aligned quadrilateral. Any given pixel of the object may "cover" multiple texels of the map, or only a fraction of one texel. For each pixel, the usual goal is to sample and filter the texture image in order to best represent the covered texel values, with a minimum of blurring or aliasing artifacts. Per-texture state variables are provided to allow the user to employ quality/performance/footprint tradeoffs in selecting how the particular texture is to be sampled.

The Texel Address Generation function of the Sampling Engine is responsible for determining how the texture maps are to be sampled. Outputs of this function include the number of texel to be fetched, along with the physical addresses of the samples and the filter weights to be applied to the samples after they are read. This information is computed given the incoming texture coordinate and gradient values, and the relevant state variables associated with the sampler and surface. This function also applies the texture coordinate address controls when converting the sample texture coordinates to map addresses.

Level of Detail Computation (Mipmapping)

Due to the specification and processing of texture coordinates at object vertices, and the subsequent object warping due to a perspective projection, the texture image may become *magnified* (where a texel covers more than one pixel) or *minified* (a pixel covers more than one texel) as it is mapped to an object. In the case where an object pixel is found to cover multiple texels (texture minification), merely choosing one (e.g., the texel sample nearest to the pixel's texture coordinate) will likely result in severe aliasing artifacts.

Mipmapping and texture filtering are techniques employed to minimize the effect of undersampling these textures. With mipmapping, software provides *mipmap levels,* a series of pre-filtered texture maps of decreasing resolutions that are stored in a fixed (monolithic) format in memory. When mipmaps are provided and enabled, and an object pixel is found to cover multiple texels (e.g., when a textured object is located a significant distance from the viewer), the device will sample the mipmap level(s) offering a texel/pixel ratio as close to 1.0 as possible.

The device supports up to 14 mipmap levels per map surface, ranging from 8192 x 8192 texels to a 1 X 1 texel. Each successive level has ½ the resolution of the previous level in the U and V directions (to a minimum of 1 texel in either direction) until a 1x1 texture map is reached. The dimensions of mipmap levels need not be a power of 2.

Each mipmap level is associated with a *Level of Detail (LOD)* number. LOD is computed as the approximate, $\log_2$ measure of the ratio of texels per pixel. The highest resolution map is considered LOD 0. A larger LOD number corresponds to lower resolution mip level.

The *Sampler[]BaseMipLevel* state variable specifies the LOD value at which the minification filter vs. the magnification filter should be applied.

When the texture map is magnified (a texel covers more than one pixel), the base map (LOD 0) texture map is accessed, and the magnification mode selects between the nearest neighbor texel or bilinear interpolation of the 4 neighboring texels on the base (LOD 0) mipmap.

Base Level Of Detail (LOD)

The per-pixel LOD is computed in an implementation-dependent manner and approximates the $\log_2$ of the texel/pixel ratio at the given pixel. The computation is typically based on the differential texel-space distances associated with a one-pixel differential distance along the screen x- and y-axes. These texel-space distances are computed by evaluating neighboring pixel texture coordinates, these coordinates being in units of texels on the base MIP level (multiplied by the corresponding surface size in texels). The q coordinates represent the third dimension for 3D (volume) surfaces, this coordinate is a constant 0 for 2D surfaces.

The ideal LOD computation is included below.

$$LOD(x,y) = \log_2[\rho(x,y)]$$

where :

$$\rho(x,y) = \max\left\{\sqrt{\left(\frac{\partial u}{\partial x}\right)^2 + \left(\frac{\partial v}{\partial x}\right)^2 + \left(\frac{\partial q}{\partial x}\right)^2}, \sqrt{\left(\frac{\partial u}{\partial y}\right)^2 + \left(\frac{\partial v}{\partial y}\right)^2 + \left(\frac{\partial q}{\partial y}\right)^2}\right\},$$

LOD Bias

A biasing offset can be applied to the computed LOD and used to artificially select a higher or lower miplevel and/or affect the weighting of the selected mipmap levels. Selecting a slightly higher mipmap level will trade off image blurring with possibly increased performance (due to better texture cache reuse). Lowering the LOD tends to sharpen the image, though at the expense of more texture aliasing artifacts.

The LOD bias is defined as sum of the *LODBias* state variable and the *pixLODBias* input from the input message (which can be non-zero only for sample_b messages). The application of LOD Bias is unconditional, therefore these variables must both be set to zero in order to prevent any undesired biasing.

Note that, while the LOD Bias is applied prior to clamping and min/mag determination and therefore can be used to control the min-vs-mag crossover point, its use has the undesired effect of actually changing the LOD used in texture filtering.

LOD Pre-Clamping

The LOD Pre-Clamping function can be enabled or disabled via the *LODPreClampEnable* state variable. Enabling pre-clamping matches OpenGL semantics .

After biasing and/or adjusting of the LOD , the computed LOD value is clamped to a range specified by the (integer and fractional bits of) *MinLOD* and *MaxLOD* state variables prior to use in Min/Mag Determination.

MaxLOD specifies the lowest resolution mip level (maximum LOD value) that can be accessed, even when lower resolution maps may be available. Note that this is the only parameter used to specify the number of valid mip levels that be can be accessed, i.e., there is no explicit "number of levels stored in memory" parameter associated with a mip-mapped texture. All mip levels from the base mip level map through the level specified by the integer bits of *MaxLOD* must be stored in memory, or operation is UNDEFINED.

MinLOD specifies the highest resolution mip level (minimum LOD value) that can be accessed, where LOD==0 corresponds to the base map. This value is primarily used to deny access to high-resolution mip levels that have been evicted from memory when memory availability is low.

MinLOD and *MaxLOD* have both integer and fractional bits. The fractional parts will limit the inter-level filter weighting of the highest or lowest (respectively) resolution map. For example if *MinLOD* is 4.5 and *MipFilter* is LINEAR, LOD 4 can contribute only up to 50% of the final texel color.

Min/Mag Determination

The biased and clamped LOD is used to determine whether the texture is being minified (scaled down) or magnified (scaled up).

The *BaseMipLevel* state variable is subtracted from the biased and clamped LOD. The *BaseMipLevel* state variable therefore has the effect of selecting the "base" mip level used to compute Min/Mag Determination. (This was added to match OpenGL semantics). Setting *BaseMipLevel* to 0 has the effect of using the highest-resolution mip level as the base map.

If the biased and clamped LOD is non-positive, the texture is being magnified, and a single (high-resolution) miplevel will be sampled and filtered using the *MagFilter* state variable. At this point the computed LOD is reset to 0.0. Note that LOD Clamping can restrict access to high-resolution miplevels.

If the biased LOD is positive, the texture is being minified. In this case the *MipFilter* state variable specifies whether one or two mip levels are to be included in the texture filtering, and how that (or those) levels are to be determined as a function of the computed LOD.

LOD Computation Pseudocode

This section illustrates the LOD biasing and clamping computation in pseudocode, encompassing the steps described in the previous sections. The computation of the initial per-pixel LOD value *LOD* is not shown.

```
Bias:       S4.8
MinLod:     U4.8
MaxLod:     U4.8
Base:       U4.1
MIPCnt:     U4
SurfMinLod: U4.8
ResMinLod:  U4.8
```

```
PerSampleMinLOD: float32

MinLod        = max(MinLod, PerSampleMinLOD)
AdjMaxLod     = min(MaxLod, MIPCnt)
AdjMinLod     = min(MinLod, MIPCnt)
AdjPR_minLOD  = ResMinLod - SurfMinLod
AdjMinLod     = max(AdjMinLod, AdjPR_minLOD)
Out_of_Bounds = AdjPR_minLOD > MIPCnt

if ( sample_b )
    LOD += Bias + bias_parameter
else if ( sample_l or ld )
    LOD = Bias + lod_parameter
else
    LOD += Bias
```

Pseudocode

```
    PreClamp = LODPreClampMode != PRECLAMP_NONE
    if ( PreClamp )
        if ( PRECLAMP_D3D )
            LOD = min(LOD, AdjMaxLod)
            LOD = max(LOD, AdjMinLod)
        else
            LOD = min(LOD, MaxLod)
            LOD = max(LOD, MinLod)
```

```
MagMode = (LOD - Base <= 0)
```

Pseudocode

```
MagClampMipNone = LODClampMagnificationMode == MAG_CLAMP_MIPNONE
```

```
if ( (MagMode && MagClampMipNone) or MipFlt == None )
    LOD = 0
    LOD = min(LOD, ceil(AdjMaxLod))
    LOD = max(LOD, floor(AdjMinLod))
else if ( MipFlt == Nearest )
```

Pseudocode

```
LOD = min(LOD, ceil(AdjMaxLod))
LOD = max(LOD, floor(AdjMinLod))
```

```
    LOD += 0.5
    LOD = floor(LOD)
else
    // MipFlt = Linear
    LOD = min(LOD, AdjMaxLod)
    LOD = max(LOD, AdjMinLod)
    TriBeta = frac(LOD)
    LOD0 = floor(LOD)
    LOD1 = LOD0 + 1

if ( ! lod )  // "LOD" message type
    Lod += SurfMinLod
```

If Out_of_Bounds is true, LOD is set to zero and instead of sampling the surface the texels are replaced with zero in all channels, except for surface formats that don't contain alpha, for which the alpha channel is replaced with one. These texels then proceed through the rest of the pipeline.

Intra-Level Filtering Setup

Depending on whether the texture is being minified or magnified, the *MinFilter* or *MagFilter* state variable (respectively) is used to select the sampling filter to be used within a mip level (intra-level, as opposed to any inter-level filter). Note that for volume maps, this selection also applies to filtering between layers.

The processing at this stage is restricted to the selection of the filter type, computation of the number and texture map coordinates of the texture samples, and the computation of any required filter parameters. The filtering of the samples occurs later on in the Sampling Engine function.

The following table summarizes the intra-level filtering modes.

Sampler[]Min/MagFilter value	Description
MAPFILTER_NEAREST	Supported on all surface types. The texel nearest to the pixel's U,V,Q coordinate is read and output from the filter.
MAPFILTER_LINEAR	Not supported on buffer surfaces. The 2, 4, or 8 texels (depending on 1D, 2D/CUBE, or 3D surface, respectively) surrounding the pixel's U,V,Q coordinate are read and a linear filter is applied to produce a single filtered texel value.
MAPFILTER_ANISOTROPIC	Not supported on buffer or 3D surfaces. A projection of the pixel onto the texture map is generated and "subpixel" samples are taken along the major axis of the projection (center axis of the longer dimension). The outermost subpixels are weighted according to closeness to the edge of the projection, inner subpixels are weighted equally. Each subpixel samples a bilinear 2x2 of texels and the results are blended according to weights to produce a filtered texel value.
MAPFILTER_MONO	Supported only on 2D surfaces. This filter is only supported with the monochrome (MONO8) surface format. The monochrome texel block of the specified size surrounding the pixel is selected and filtered.

MAPFILTER_NEAREST

When the MAPFILTER_NEAREST is selected, the texel with coordinates nearest to the pixel's texture coordinate is selected and output as the single texel sample coordinates for the level. This is a form of "Point Sampling".

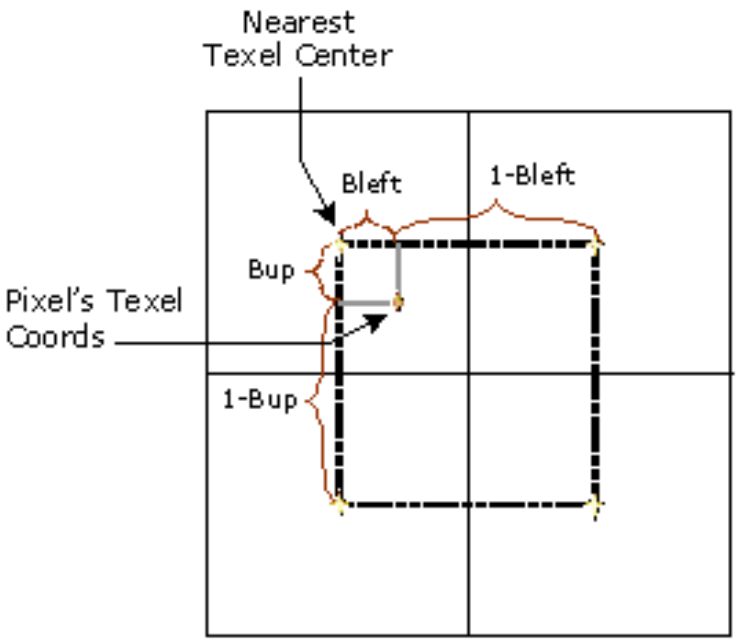
Corner Texel Mode

MAPFILTER_LINEAR

The following description indicates behavior of the MIPFILTER_LINEAR filter for 2D and CUBE surfaces. 1D and 3D surfaces follow a similar method but with a different number of dimensions available.

When the MAPFILTER_LINEAR filter is selected on a 2D surface, the 2x2 region of texels surrounding the pixel's texture coordinate are sampled and later bilinearly filtered. The filter weights each texel's contribution according to its distance from the pixel center. Texels further from the pixel center receive a smaller weight.

Bilinear Filter Sampling

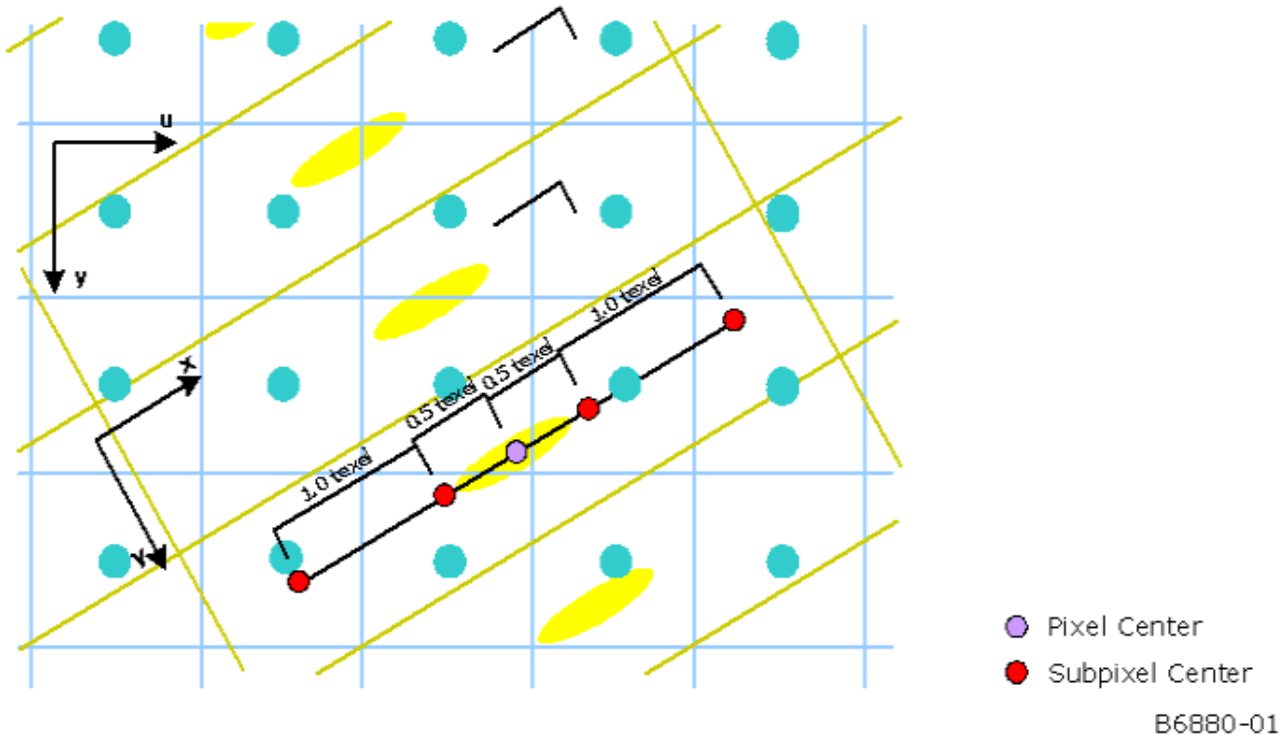


B6879-01

MAPFILTER_ANISOTROPIC

The MAPFILTER_ANISOTROPIC texture filter attempts to compensate for the anisotropic mapping of pixels into texture map space. A possibly non-square set of texel sample locations will be sampled and later filtered. The *MaxAnisotropy* state variable is used to select the maximum aspect ratio of the filter employed, up to 16:1.

The algorithm employed first computes the major and minor axes of the pixel projection onto the texture map. LOD is chosen based on the minor axis length in texel space. The anisotropic "ratio" is equal to the ratio between the major axis length and the minor axis length. The next larger even integer above the ratio determines the anisotropic number of "ways", which determines how many subpixels are chosen. A line along the major axis is determined, and "subpixels" are chosen along this line, spaced one texel apart, as shown in the diagram below. In this diagram, the texels are shown in light blue, and the pixels are in yellow.



Each subpixel samples a bilinear 2x2 around it just as if it was a single pixel. The result of each subpixel is then blended together using equal weights on all interior subpixels (not including the two endpoint subpixels). The endpoint subpixels have lesser weight, the value of which depends on how close the "ratio" is to the number of "ways". This is done to ensure continuous behavior in animation.

MAPFILTER_MONO

When the MAPFILTER_MONO filter is selected, a block of monochrome texels surrounding the pixel sample location are read and filtered using the kernel described below. The size of this block is controlled by **Monochrome Filter Height** and **Width** (referred to here as N_v and N_u, respectively) state. Filters from 1x1 to 7x7 are supported (not necessarily square).

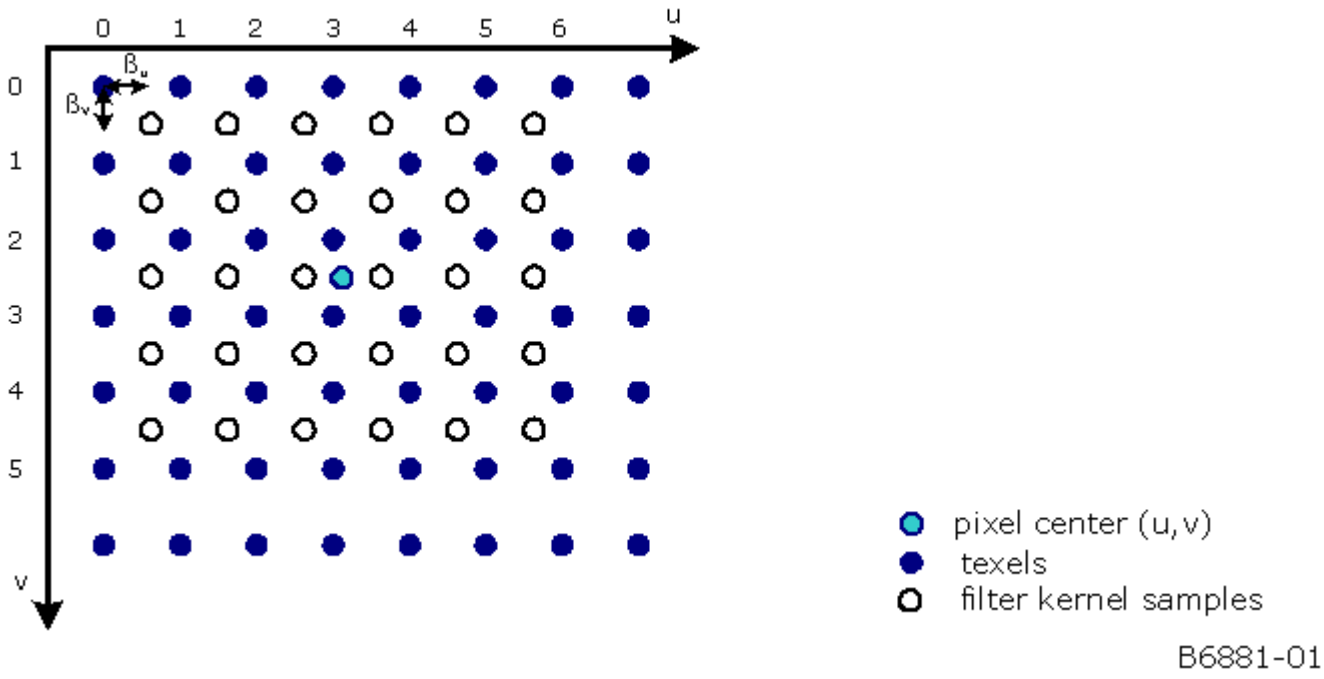
The figure below shows a 6x5 filter kernel as an example. The footprint of the filter (filter kernel samples) is equal to the size of the filter and the pixel center lies at the exact center of this footprint. The position of the upper left filter kernel sample (u_f, v_f) relative to the pixel center at (u, v) is given by the following:

$$u_f = u - \frac{N_u}{2}$$

$$v_f = v - \frac{N_v}{2}$$

b_u and b_v are the fractional parts of u_f and v_f, respectively. The integer parts select the upper left texel for the kernel filter, given here as $T_{0,0}$.

Sampling Using MAPFILTER_MONO



The formula for the final filter output F is given by the following. Since this is a monochrome filter, each texel value (T) is a single bit, and the output F is an intensity value that is replicated across the color and alpha channels.

$$S = \frac{1}{N_u * N_v}$$

$$F = \left[(1-\beta_u)(1-\beta_v)\sum_{i=0}^{N_u-1}\sum_{j=0}^{N_v-1} T_{i,j} + \beta_u(1-\beta_v)\sum_{i=1}^{N_u}\sum_{j=0}^{N_v-1} T_{i,j} + (1-\beta_u)\beta_v\sum_{i=0}^{N_u-1}\sum_{j=1}^{N_v} T_{i,j} + \beta_u\beta_v\sum_{i=1}^{N_u}\sum_{j=1}^{N_v} T_{i,j} \right] * S$$

Inter-Level Filtering Setup

The *MipFilter* state variable determines if and how texture mip maps are to be used and combined. The following table describes the various mip filter modes:

MipFilter Value	Description
MIPFILTER_NONE	Mipmapping is DISABLED. Apply a single filter on the highest resolution map available (after LOD clamping).
MIPFILTER_NEAREST	Choose the nearest mipmap level and apply a single filter to it. Here the biased LOD will be rounded to the nearest integer to obtain the desired miplevel. LOD Clamping may further restrict this miplevel selection.
MIPFILTER_LINEAR	Apply a filter on the two closest mip levels and linear blend the results using the distance between the computed LOD and the level LODs as the blend factor. Again, LOD Clamping may further restrict the selection of miplevels (and the blend factor between them).

When minifying and MIPFILTER_NEAREST is selected, the computed LOD is rounded to the nearest mip level.

When minifying and MIPFILTER_LINEAR is selected, the fractional bits of the computed LOD are used to generate an inter-level blend factor. The LOD is then truncated. The mip level selected by the truncated LOD, and the next higher (lower resolution) mip level are determined.

Regardless of *MipFilter* and the min/mag determination, all computed LOD values (two for MIPFILTER_LINEAR, otherwise one) are then unconditionally clamped to the range specified by the (integer bits of) *MinLOD* and *MaxLOD* state variables.

Texture Address Control

The *[TCX,TCY,TCZ]ControlMode* state variables control the access and/or generation of texel data when the specific texture coordinate component falls *outside* of the normalized texture map coordinate range [0,1).

The table below provides all the supported Address Control modes for each direction.

TC[X,Y,Z] Control	Operation
TEXCOORDMODE_CLAMP	Clamp to the texel value at the edge of the map.
TEXCOORDMODE_CLAMP_BORDER	Use the texture map's border color for any texel samples falling outside the map. The border color is specified via a pointer in SAMPLER_STATE.
TEXCOORDMODE_HALF_BORDER	Similar to CLAMP_BORDER except texels outside of the map are clamped to a value halfway between the edge texel and the border color.
TEXCOORDMODE_WRAP	Upon crossing an edge of the map, repeat at the other side of the map in the same dimension.
TEXCOORDMODE_CUBE	Only used for cube maps. Here texels from adjacent cube faces can be sampled along the edges of faces. This is considered the highest quality mode for cube environment maps.
TEXCOORDMODE_MIRROR	Similar to the wrap mode, though reverse direction through the map each time an edge is crossed. INVALID for use with unnormalized texture coordinates.
TEXCOORDMODE_MIRROR_ONCE	Similar to the wrap mode, though reverse direction through the map each time an edge is crossed. INVALID for use with unnormalized texture coordinates.

Separate controls are provided for texture TCX, TCY, TCZ coordinate components so, for example, the TCX coordinate can be wrapped while the TCY coordinate is clamped. Note that there are no controls provided for the TCW component as it is only used to scale the other 3 components before addressing modes are applied.

Programming Note	
Context:	Texture Address Control
TEXCOORDMODE_CUBE can only be used with SURFTYPE_CUBE	

Maximum Wraps/Mirrors

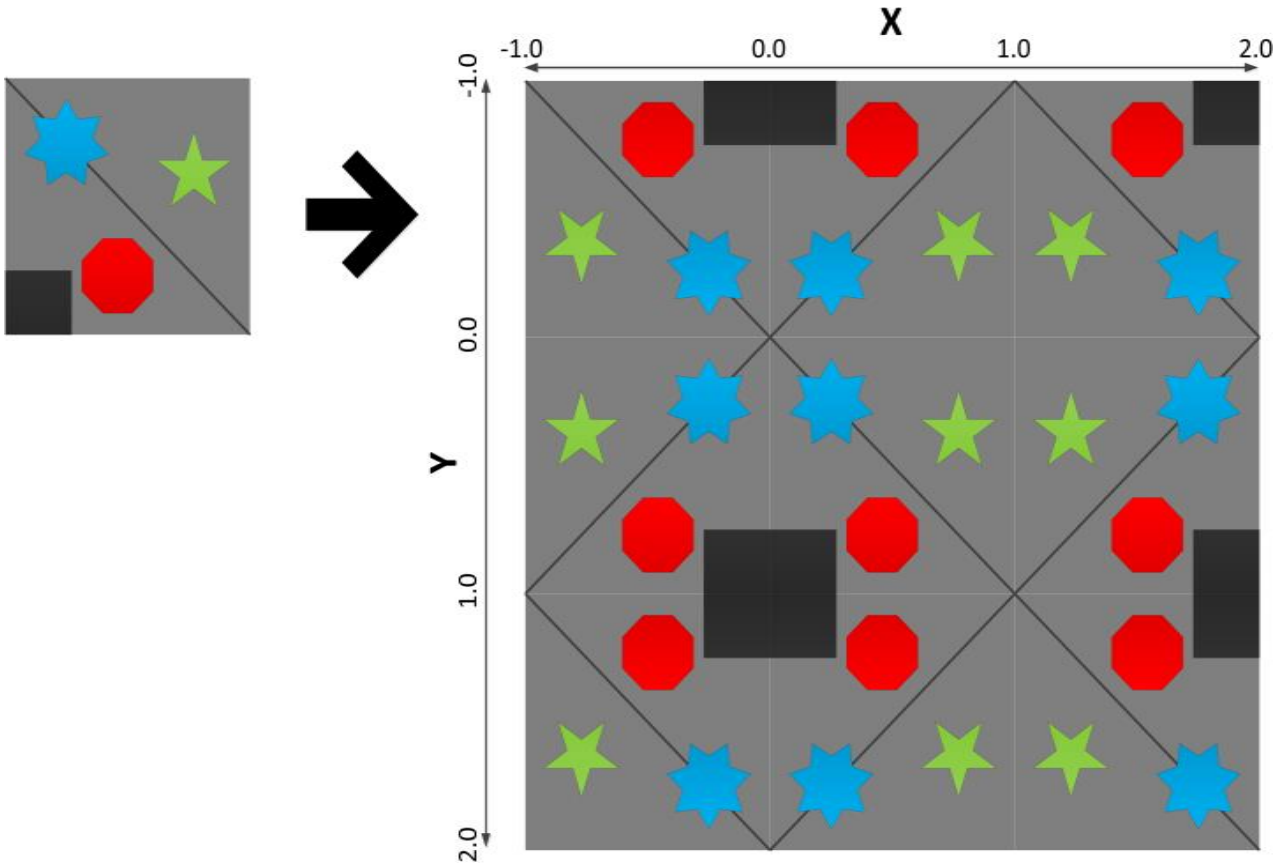
The number of map wraps on a given object is limited to 32. Going beyond this limit is legal, but may result in artifacts due to insufficient internal precision, especially evident with larger surfaces. Precision loss starts at the subtexel level (slight color inaccuracies) and eventually reaches the texel level (choosing the wrong texels for filtering).

Note: For **Wrap Shortest** mode, the setup kernel has already taken care of correctly interpolating the texture coordinates. Software needs to specify TEXCOORDMODE_WRAP mode for the sampler that is provided with wrap-shortest texture coordinates, or artifacts may be generated along map edges.

TEXCOORDMODE_MIRROR Mode

TEXCOORDMODE_MIRROR addressing mode is similar to Wrap mode, though here the base map is flipped at every integer junction. For example, for U values between 0 and 1, the texture is addressed normally, between 1 and 2 the texture is flipped (mirrored), between 2 and 3 the texture is normal again, and so on. The second row of pictures in the figure below indicate a map that is mirrored in one direction and then both directions. You can see that in the mirror mode every other integer map wrap the base map is mirrored in either direction.

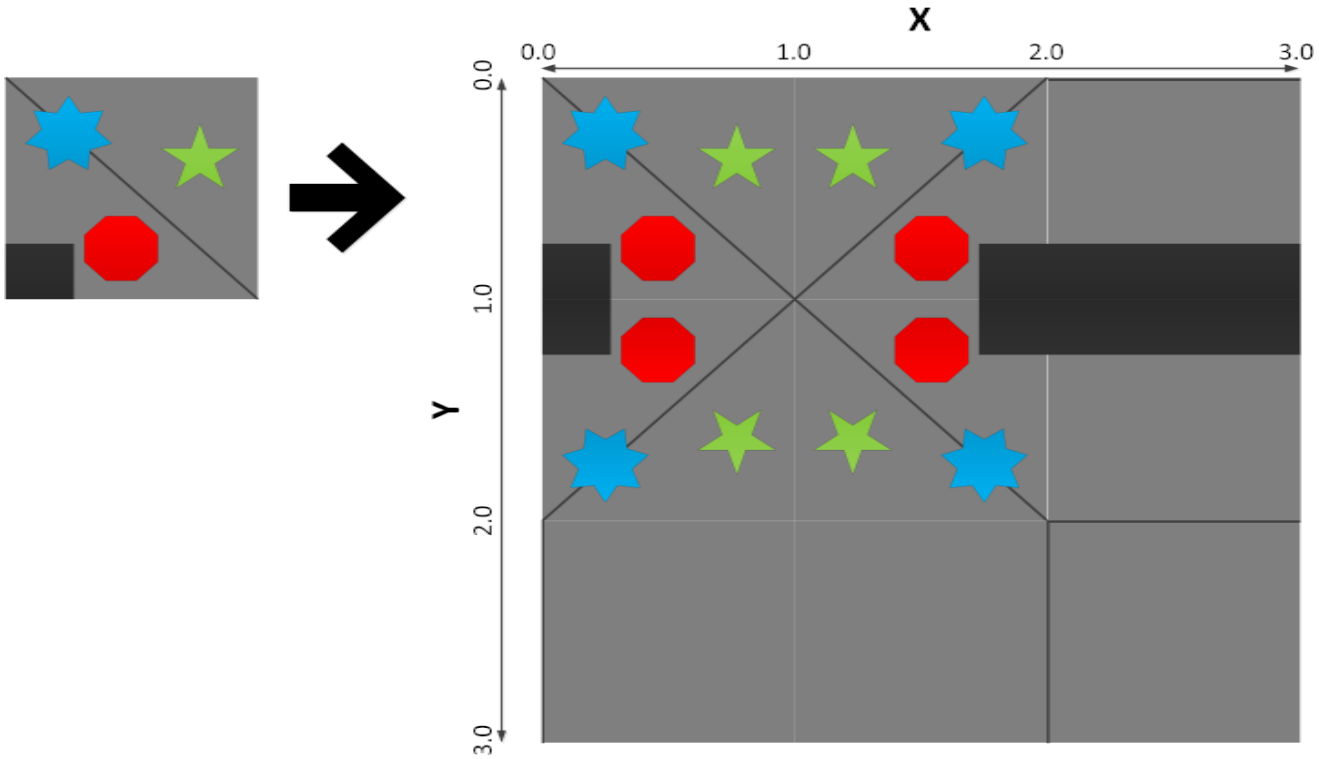
The example below shows how a simple 2D map with TEXCOORDMODE_MIRROR for both TCX and TCY is mapped.



TEXCOORDMODE_MIRROR_ONCE Mode

The TEXCOORDMODE_MIRROR_ONCE addressing mode is a combination of Mirror and Clamp modes. The absolute value of the texture coordinate component is first taken (thus mirroring about 0), and then the result is clamped to 1.0. The map is therefore mirrored once about the origin, and then clamped thereafter. This mode is used to reduce the storage required for symmetric maps.

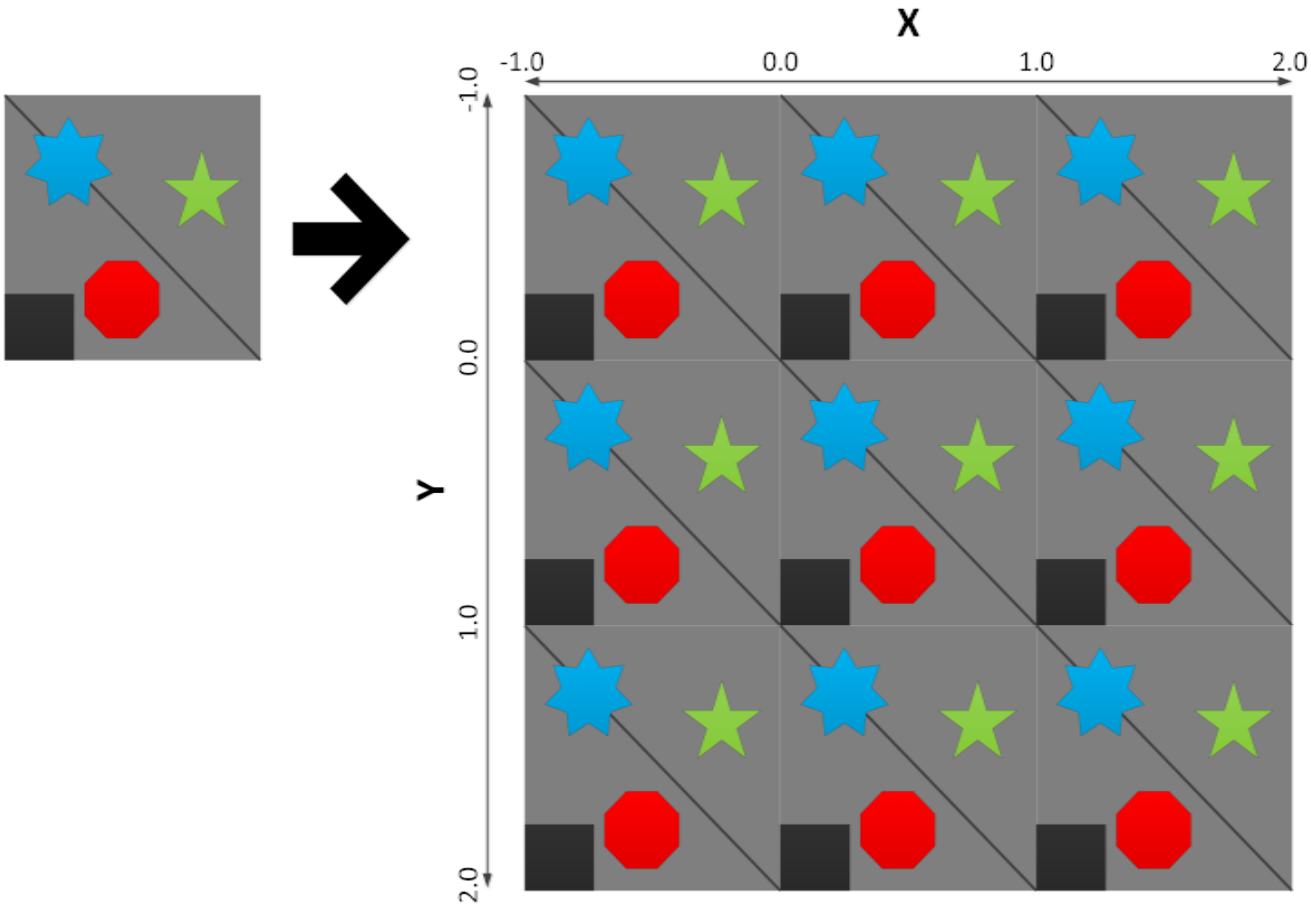
The example below shows how a simple 2D map with TEXCOORDMODE_MIRROR_ONCE for both TCX and TCY is mapped.



TEXCOORDMODE_WRAP Mode

In TEXCOORDMODE_WRAP addressing mode, the integer part of the texture coordinate is discarded, leaving only a fractional coordinate value. This results in the effect of the base map ([0,1)) being continuously repeated in all (axes-aligned) directions. Note that the interpolation between coordinate values 0.1 and 0.9 passes through 0.5 (as opposed to WrapShortest mode which interpolates through 0.0).

The example below shows how a simple 2D map with TEXCOORDMODE_WRAP for both TCX and TCY is mapped.

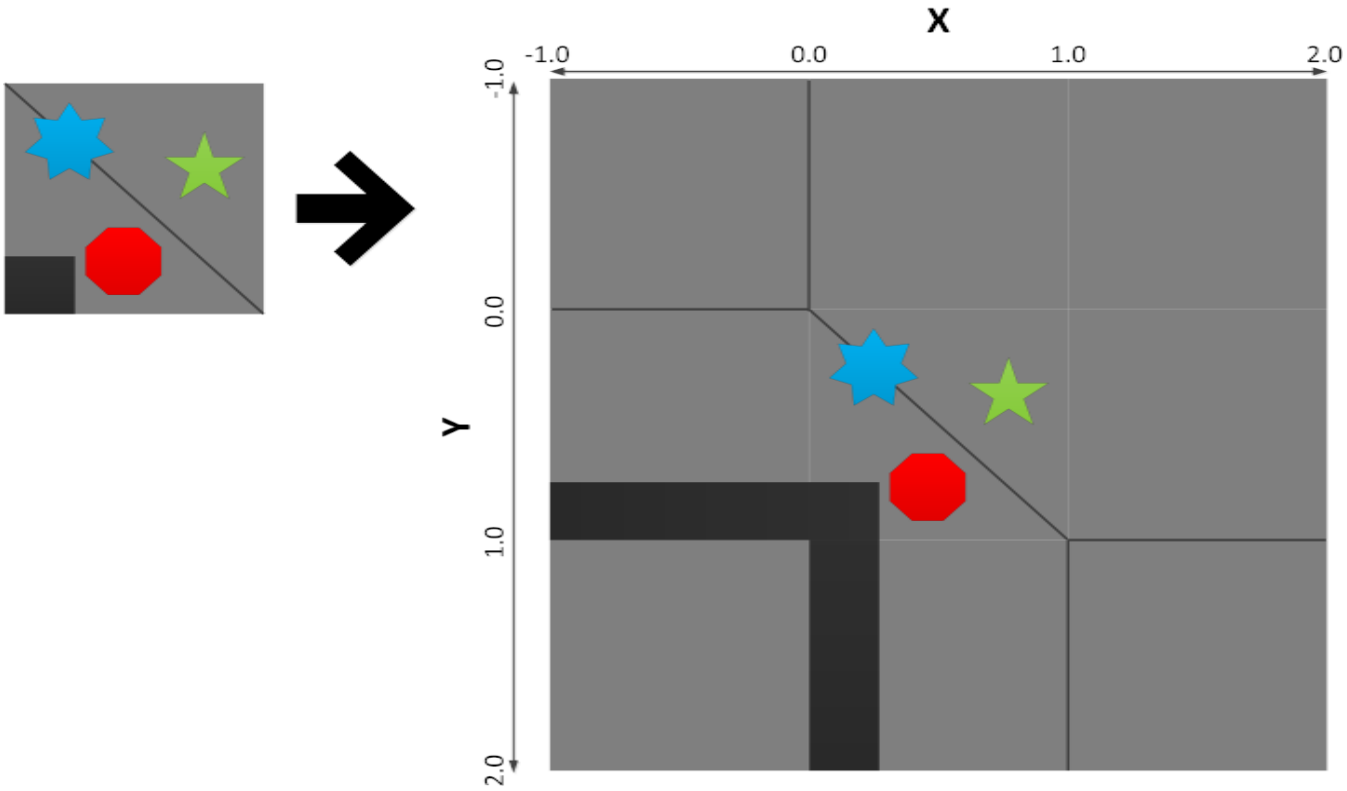


TEXCOORDMODE_CLAMP Mode

The TEXCOORDMODE_CLAMP addressing mode repeats the “edge” texel when the texture coordinate extends outside the [0,1) range of the base texture map. This is contrasted to TEXCOORDMODE_CLAMPBORDER mode which defines a separate texel value for off-map samples. TEXCOORDMODE_CLAMP is also supported for cube maps, where texture samples will only be obtained from the intersecting face (even along edges).

The figure below illustrates the effect of clamp mode. The base texture map is shown, along with a texture mapped object with texture coordinates extending outside of the base map region.

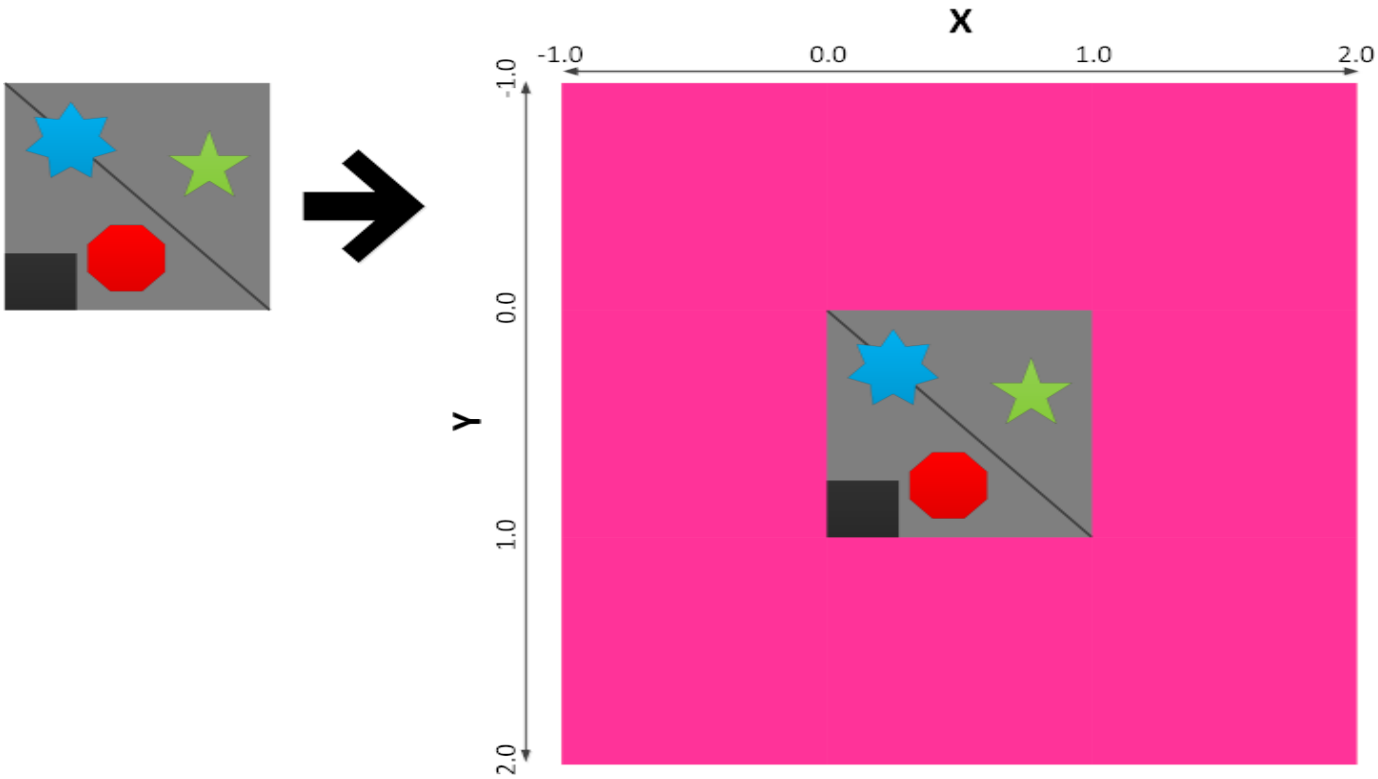
Texture Clamp Mode



TEXCOORDMODE_CLAMPBORDER Mode

For non-cube map textures, TEXCOORDMODE_CLAMPBORDER addressing mode specifies that the texture map's border value *BorderColor* is to be used for any texel samples that fall outside of the base map. The border color is specified via a pointer in SAMPLER_STATE.

The example below shows how a simple 2D map with TEXCOORDMODE_CLAMPBORDER for both TCX and TCY is mapped.



TEXCOORDMODE_CUBE Mode

For cube map textures TEXCOORDMODE_CUBE addressing mode can be set to allow inter-face filtering. When texel sample coordinates that extend beyond the selected cube face (e.g., due to intra-level filtering near a cube edge), the correct sample coordinates on the adjoining face will be computed. This will eliminate artifacts along the cube edges, though some artifacts at cube corners may still be present.

Texel Fetch

The Texel Fetch function of the Sampling Engine reads the texture map contents specified by the texture addresses associated with each texel sample. The texture data is read either directly from the memory-resident texture map, or from internal texture caches. The texture caches can be invalidated by the **Sampler Cache Invalidate** field of the MI_FLUSH instruction or via the **Read Cache Flush Enable** bit of PIPE_CONTROL. Except for consideration of coherency with CPU writes to textures and rendered textures, the texture cache does not affect the functional operation of the Sampling Engine pipeline.

When the surface format of a texture is defined as being a compressed surface, the Sampler will automatically decompress from the stored format into the appropriate [A]RGB values. The compressed texture storage formats and decompression algorithms can be found in the *Memory Data Formats* chapter. When the surface format of a texture is defined as being an index into the texture palette (format names includiong "Px"), the palette lookup of the index determines the appropriate RGB values.

Texel Chroma Keying

ChromaKey is a term used to describe a method of effectively removing or replacing a specific range of texel values from a map that is applied to a primitive, e.g., in order to define transparent regions in an RGB map. The Texel Chroma Keying function of the Sampling Engine pipeline conditionally tests texel samples against a "key" range, and takes certain actions if any texel samples are found to match the key.

Chroma Key Testing

ChromaKey refers to testing the texel sample components to see if they fall within a range of texel values, as defined by *ChromaKey[][High,Low]* state variables. If each component of a texel sample is found to lie within the respective (inclusive) range and ChromaKey is enabled, then an action will be taken to remove this contribution to the resulting texel stream output. Comparison is done separately on each of the channels and only if all 4 channels are within range the texel will be eliminated.

The Chroma Keying function is enabled on a per-sampler basis by the *ChromaKeyEnable* state variable.

The *ChromaKey[][High,Low]* state variables define the tested color range for a particular texture map.

Chroma Key Effects

There are two operations that can be performed to "remove" matching texel samples from the image. The *ChromaKeyEnable* state variable must first enable the chroma key function. The *ChromaKeyMode* state variable then specifies which operation to perform on a per-sampler basis.

The *ChromaKeyMode* state variable has the following two possible values:

KEYFILTER_KILL_ON_ANY_MATCH: Kill the pixel if any contributing texel sample matches the key.

KEYFILTER_REPLACE_BLACK: Here the sample is replaced with (0,0,0,0).

The Kill Pixel operation has an effect on a pixel only if the associated sampler is referenced by a sample instruction in the pixel shader program. If the sampler is not referenced, the chroma key compare is not done and pixels cannot be killed based on it.

Shadow Prefilter Compare

When a *sample_c* message type is processed, a special shadow-mapping precomparison is performed on the texture sample values prior to filtering. Specifically, each texture sample value is compared to the "ref" component of the input message, using a compare function selected by *ShadowFunction*, and described in the table below. Note that only single-channel texel formats are supported for shadow mapping, and so there is no specific color channel on which the comparison occurs.

ShadowFunction	Result
PREFILTEROP_ALWAYS	0.0
PREFILTEROP_NEVER	1.0
PREFILTEROP_LESS	(texel < ref) ? 0.0 : 1.0
PREFILTEROP_EQUAL	(texel == ref) ? 0.0 : 1.0
PREFILTEROP_LEQUAL	(texel <= ref) ? 0.0 : 1.0
PREFILTEROP_GREATER	(texel > ref) ? 0.0 : 1.0
PREFILTEROP_NOTEQUAL	(texel != ref) ? 0.0 : 1.0
PREFILTEROP_GEQUAL	(texel >= ref) ? 0.0 : 1.0

The binary result of each comparison is fed into the subsequent texture filter operation (in place of the texel's value which would normally be used).

Software is responsible for programming the "ref" component of the input message such that it approximates the same distance metric programmed in the texture map (e.g., distance from a specific light to the object pixel). In this way, the comparison function can be used to generate "in shadow" status for each texture sample, and the filtering operation can be used to provide soft shadow edges.

Texel Filtering

The Texel Filtering function of the Sampling Engine performs any required filtering of multiple texel values on and possibly between texture map layers and levels. The output of this function is a single texel color value.

The state variables *MinFilter*, *MagFilter*, and *MipFilter* are used to control the filtering of texel values. The *MipFilter* state variable specifies how many mipmap levels are included in the filter, and how the results of any filtering on these separate levels are combined to produce a final texel color. The *MinFilter* and *MagFilter* state variables specify how texel samples are filtered within a level.

Texel Color Gamma Linearization

This function is supported to allow pre-gamma-corrected texel RGB (not A) colors to be mapped back into linear (gamma=1.0) gamma space prior to (possible) blending with, and writing to the Color Buffer. This permits higher quality image blending by performing the blending on colors in linear gamma space.

This function is enabled on a per-texture basis by use of a surface format with "_SRGB" in its name. If enabled, the pre-filtered texel RGB color to be converted to gamma=1.0 space by applying a ^(2.4) exponential function.

Multisampled Surface Behavior

The ld message has added an additional parameter for sample index (si) to support unfiltered loading from a multisampled surface.

The sampleinfo message returns specific parameters associated with a multisample surface. The resinfo message returns the height, width, depth, and MIP count of the surface (in units of *pixels*, not samples).

Any of the other messages (sample*, LOD, load4) used with a (4x) multisampled surface would sample a surface with double the height and width as indicated in the surface state. Each pixel position on the original-sized surface is replaced with 2x2 samples that have the following arrangement:

sample 0	sample 2
sample 1	sample 3

This behavior is useful when implementing the multisample resolve operation by selecting MAPFILTER_LINEAR and rendering a full-screen rectangle half the size in each dimension of the source texture map (multisampled surface). If pixel offsets are set correctly, each pixel is the average of the four underlying samples.

Multisample Control Surface

Three new messages have been defined for the sampling engine, *ld_mcs*, *ld2dms*, and *ld2dss*. A pixel shader kernel sampling from an multisampled surface using an MCS must first sample from the MCS surface using the *ld_mcs* message. This message behaves like the *ld* message, except that the surface is defined by the MCS fields of SURFACE_STATE rather than the normal fields. The surface format is effectively R8_UINT for 4x surfaces and R32_UINT for 8x surfaces, thus data is returned in unsigned integer format. Following the *ld_mcs*, the kernel issues a *ld2dms* message to sample the surface itself. The integer value from the MCS surface is delivered in the mcs parameter of this messages.

Since *sample* is no longer supported on multisampled surfaces, the multisample resolve must be done using *ld2dms*. For surfaces with **Multisampled Surface Storage Format** set to MSFMT_MSS and **MCS Enable** set to enabled, an optimization is available to enable higher performance for compressed pixels. The *ld2dss* message can be used to sample from a particular sample slice on the surface. By examining the MCS value, software can determine which sample slices to sample from. A simple optimization with probable large return in performance is to compare the MCS value to zero (indicating all samples are on sample slice 0), and sample only from sample slice 0 using *ld2dss* if MCS is zero. Sample slice 0 is the pixel color in this case. If MCS is not zero, each sample is then obtained using *ld2dms* messages and the results are averaged in the kernel after being returned. Refer to the multisample storage format in the GPU Overview volume for more details.

State

BINDING_TABLE_STATE

SW Generated BINDING_TABLE_STATE

HW Generated BINDING_TABLE_STATE

For **SAMPLER_STATE** for **Sample_8X8** see 3D-Media-GPGPU Engine > Shared Functions > Media Sampler > Sample_8x8 State > SAMPLER_STATE

The 3D sampler uses both surface state objects (RENDER_SURFACE_STATE) as well as sampler state objects (SAMPLER_STATE). These objects are cached locally in the sampler state cache for improved performance as it is assumed that many sampler messages will utilize the same surface and sampler states.

Programming Note	
Context:	Out of Bounds Handling
If a pointer to sampler or surface state goes beyond the end of the sampler or surface state buffer (as defined by the associated size field of the STATE_BASE_ADDRESS command) the sampler will force the address offset to cache-line 0 from the defined Base Address. The result of this state fetch is undefined and depends on how the state buffer has been populated.	

Surface State Fetch

Surface state is fetched from system memory using a Binding Table Pointer (**BTP**). The **BTP** is a 16-bit value provided by the command stream (not directly by the shader) which determines the binding-table to be used. An 8-bit Binding Table Index (**BTI**) is then provided by the shader via the message descriptor, which indicates the offset into the Binding Table. The BTP and BTI are relative to the **Surface State Base Address** and the binding table itself resides in system memory. The contents of the Binding Table is a list of pointers to surface state objects. The pointer from the Binding Table is also relative to the **Sampler State Base Address,** and points directly to a 256-bit **RENDER_SURFACE_STATE** object which sampler will fetch and store in its internal state cache.

For Positional Shaing (POSH) there is a separate set of base addresses for Bindless and Non-Bindless Surface State called POSH Surface State Offset and POSH Bindless Surface State Base Address. They work exactly the same, but allow different base addresses for positional shading.

Sampler State Fetch

SAMPLER_STATE objects are fetched independely of surface state and cached locally in the 3D sampler independently (there may one or more SAMPLER_STATE objects associates with one or more RENDER_SURFACE_STATE objects). The sampler state is fetched using the **Sampler State Pointer (SSP)** which is provided either in the message header or directly from the command stream (message headers are not required). The **SSP** is an offset relative to the **Dynamic_State_Base_Address** and selects a table of 16 sampler states. The 4-bit **Sampler Index** (**SI**) in the message descriptor is used to select the specific **SAMPLER_STATE** object to be fetched from system memory and cached locally in the 3D sampler.

For Positional Shading (POSH) there is a separate set of base addresses available for fetching sampler state called **POSH Bindless Surface State Base Address** and **POSH Dynamic State Base Address.**

State Caching

As mentioned above, the 3D Sampler allows for automatic caching of **RENDER_SURFACE_STATE** objects and **SAMPLER_STATE** objects to provide higher performance. Coherency with system memory in the state cache, like the texture cache is handled partially by software. It is expected that the command stream or shader will issue Cache Flush operation or Cache_Flush sampler message to ensure that the L1 cache remains coherent with system memory.

Programming Note	
Context:	State Cache Coherency
Whenever the value of the Dynamic_State_Base_Addr, Surface_State_Base_Addr are altered, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory. Whenever the RENDER_SURFACE_STATE object in memory pointed to by the **Binding Table Pointer (BTP)** and **Binding Table Index (BTI)** is modified or SAMPLER_STATE object pointed to by the **Sampler State Pointer (SSP)** and **Sampler Index (SI)** is modified, the L1 state cache must be invalidated to ensure the new surface or sampler state is fetched from system memory.	

SURFACE_STATE

The surface state is stored as individual elements, each with its own pointer in the binding table or its own entry in a memory heap in memory. Each surface state element is aligned to a 32-byte boundary.

Surface state defines the state needed for the following objects:

- texture maps (1D, 2D, 3D, cube) read by the sampling engine
- buffers read by the sampling engine
- constant buffers read by the data cache via the data port
- render targets read/written by the render cache via the data port
- streamed vertex buffer output written by the render cache via the data port
- media surfaces read from the texture cache or render cache via the data port
- media surfaces written to the render cache via the data port

The surface state definition can be found in the following section:

RENDER_SURFACE_STATE

Surface Formats

The RENDER_SURFACE_STATE contains a 9-bit field called **Surface Format**, which defines the exact format of the surface being sampled. The definition of the encodings for each supported format, including compressed formats can be found in the following section:

SURFACE_FORMAT

For ASTC formats, the **ASTC Enable** bit in the RENDER_SURFACE_STATE must be set to 1. When set, the definition of the 9-bit **Surface Format** changes. This table describes all supported formats for block based ASTC textures.

SURFACE_FORMAT for All ASTC Formats

Value	[8] LDR/Full [7] 2D/3D [6] U8srgb /FLT16	Width 2D [5:3] 3D [5:4]	Height 2D [2:0] 3D [3:2]	Depth 2D: N/A 3D: [1:0]	Binary Form	Name	(BPE)
000h	000	0	0		000 000 000	ASTC_LDR_2D_4x4_U8sRGB	8.00
008h	000	1	0		000 001 000	ASTC_LDR_2D_5x4_U8sRGB	6.40
009h	000	1	1		000 001 001	ASTC_LDR_2D_5x5_ U8sRGB	5.12
011h	000	2	1		000 010 001	ASTC_LDR_2D_6x5_ U8sRGB	4.27
012h	000	2	2		000 010 010	ASTC_LDR_2D_6x6_ U8sRGB	3.56
021h	000	4	1		000 100 001	ASTC_LDR_2D_8x5_ U8sRGB	3.20
022h	000	4	2		000 100 010	ASTC_LDR_2D_8x6_ U8sRGB	2.67
031h	000	6	1		000 110 001	ASTC_LDR_2D_10x5_ U8sRGB	2.56
032h	000	6	2		000 110 010	ASTC_LDR_2D_10x6_ U8sRGB	2.13
024h	000	4	4		000 100 100	ASTC_LDR_2D_8x8_ U8sRGB	2.00
034h	000	6	4		000 110 100	ASTC_LDR_2D_10x8_ U8sRGB	1.60
036h	000	6	6		000 110 110	ASTC_LDR_2D_10x10_ U8sRGB	1.28
03eh	000	7	6		000 111 110	ASTC_LDR_2D_12x10_ U8sRGB	1.07
03fh	000	7	7		000 111 111	ASTC_LDR_2D_12x12_ U8sRGB	0.89
040h	001	0	0		001 000 000	ASTC_LDR_2D_4x4_FLT16	8.00
048h	001	1	0		001 001 000	ASTC_LDR_2D_5x4_FLT16	6.40
049h	001	1	1		001 001 001	ASTC_LDR_2D_5x5_FLT16	5.12
051h	001	2	1		001 010 001	ASTC_LDR_2D_6x5_FLT16	4.27
052h	001	2	2		001 010 010	ASTC_LDR_2D_6x6_FLT16	3.56
061h	001	4	1		001 100 001	ASTC_LDR_2D_8x5_FLT16	3.20
062h	001	4	2		001 100 010	ASTC_LDR_2D_8x6_FLT16	2.67
071h	001	6	1		001 110 001	ASTC_LDR_2D_10x5_FLT16	2.56
072h	001	6	2		001 110 010	ASTC_LDR_2D_10x6_FLT16	2.13
064h	001	4	4		001 100 100	ASTC_LDR_2D_8x8_FLT16	2.00
074h	001	6	4		001 110 100	ASTC_LDR_2D_10x8_FLT16	1.60
076h	001	6	6		001 110 110	ASTC_LDR_2D_10x10_FLT16	1.28
07eh	001	7	6		001 111 110	ASTC_LDR_2D_12x10_FLT16	1.07
07fh	001	7	7		001 111 111	ASTC_LDR_2D_12x12_FLT16	0.89

Notes and Workarounds

Programming Note	
Context:	SURFACE_STATE
• ASTC_LDR_3D* surface types are not supported. • ASTC_FULL_* surface types are not supported.	

Programming Note	
Context:	ASTC with Multiple LODs
If sampling an ASTC surface with block size of 5X5 or 5X4 and 0 < [(max(Suface_Width»1,1)%10] < 6, and accessing LOD=2 or higher, then the sampled texels will be shifted to the left by 1.	

Sampler Output Channel Mapping

The following table indicates the mapping of the channels from the surface to the channels output from the sampling engine. Formats with all four channels (R/G/B/A) in their name map each surface channel to the corresponding output, thus those formats are not shown in this table.

There are further restrictions listed after the table below on the use of specific surfaces.

Some formats are supported only in DX10/OGL **Border Color Mode**. Those formats have only that mode indicated. Formats that behave the same way in both **Border Color Modes** are indicated by that column being blank. See the programming notes below the following table for more details on how to support these surfaces.

Surface Format Name	Filtering	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A
R32G32B32A32_FLOAT					R	G	B	A
R32G32B32A32_SINT				DX10/OGL	R	G	B	A
R32G32B32A32_UINT				DX10/OGL	R	G	B	A
R32G32B32X32_FLOAT					R	G	B	1.0
R32G32B32_FLOAT					R	G	B	1.0
R32G32B32_SINT				DX10/OGL	R	G	B	1.0
R32G32B32_UINT				DX10/OGL	R	G	B	1.0
R16G16B16A16_UNORM					R	G	B	A
R16G16B16A16_SNORM					R	G	B	A
R16G16B16A16_SINT				DX10/OGL	R	G	B	A
R16G16B16A16_UINT				DX10/OGL	R	G	B	A
R16G16B16A16_FLOAT					R	G	B	A
R32G32_FLOAT				DX10/OGL	R	G	0.0	1.0
R32G32_SINT				DX10/OGL	R	G	0.0	1.0
R32G32_UINT				DX10/OGL	R	G	0.0	1.0

Surface Format Name	Filtering	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A
R32_FLOAT_X8X24_TYPELESS		Yes		DX10/OGL	R	0.0	0.0	1.0
X32_TYPELESS_G8X24_UINT				DX10/OGL	0.0	G	0.0	1.0
L32A32_FLOAT				DX10/OGL	L	L	L	A
R16G16B16X16_UNORM					R	G	B	1.0
R16G16B16X16_FLOAT					R	G	B	1.0
A32X32_FLOAT					0.0	0.0	0.0	A
L32X32_FLOAT					L	L	L	1.0
I32X32_FLOAT					I	I	I	I
B8G8R8A8_UNORM			Yes		R	G	B	A
B8G8R8A8_UNORM_SRGB					R	G	B	A
R10G10B10A2_UNORM					R	G	B	A
R10G10B10A2_UNORM_SRGB					R	G	B	A
R10G10B10A2_UINT				DX10/OGL	R	G	B	A
R10G10B10_SNORM_A2_UNORM					R	G	B	A
R8G8B8A8_UNORM					R	G	B	A
R8G8B8A8_UNORM_SRGB					R	G	B	A
R8G8B8A8_SNORM					R	G	B	A
R8G8B8A8_SINT				DX10/OGL	R	G	B	A
R8G8B8A8_UINT				DX10/OGL	R	G	B	A
R16G16_UNORM				DX10/OGL	R	G	0.0	1.0
R16G16_SNORM				DX10/OGL	R	G	0.0	1.0
R16G16_SINT				DX10/OGL	R	G	0.0	1.0
R16G16_UINT				DX10/OGL	R	G	0.0	1.0
R16G16_FLOAT				DX10/OGL	R	G	0.0	1.0
B10G10R10A2_UNORM					R	G	B	A
B10G10R10A2_UNORM_SRGB					R	G	B	A
R11G11B10_FLOAT					R	G	B	1.0
R32_SINT				DX10/OGL	R	0.0	0.0	1.0
R32_UINT				DX10/OGL	R	0.0	0.0	1.0
R32_FLOAT		Yes		DX10/OGL	R	0.0	0.0	1.0
R24_UNORM_X8_TYPELESS		Yes		DX10/OGL	R	0.0	0.0	1.0
X24_TYPELESS_G8_UINT				DX10/OGL	0.0	G	0.0	1.0
L16A16_UNORM					L	L	L	A
I24X8_UNORM		Yes			I	I	I	I
L24X8_UNORM		Yes			L	L	L	1.0

Surface Format Name	Filtering	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A
A24X8_UNORM		Yes			0.0	0.0	0.0	A
I32_FLOAT		Yes			I	I	I	I
L32_FLOAT		Yes			L	L	L	1.0
A32_FLOAT		Yes			0.0	0.0	0.0	A
B8G8R8X8_UNORM			Yes		R	G	B	1.0
B8G8R8X8_UNORM_SRGB					R	G	B	1.0
R8G8B8X8_UNORM					R	G	B	1.0
R8G8B8X8_UNORM_SRGB					R	G	B	1.0
R9G9B9E5_SHAREDEXP					R	G	B	1.0
B10G10R10X2_UNORM					R	G	B	1.0
L16A16_FLOAT					L	L	L	A
B5G6R5_UNORM			Yes		R	G	B	1.0
B5G6R5_UNORM_SRGB					R	G	B	1.0
B5G5R5A1_UNORM			Yes		R	G	B	A
B5G5R5A1_UNORM_SRGB					R	G	B	A
B4G4R4A4_UNORM			Yes		R	G	B	A
B4G4R4A4_UNORM_SRGB					R	G	B	A
R8G8_UNORM				DX10/OGL	R	G	0.0	1.0
R8G8_SNORM			Yes	DX10/OGL	R	G	0.0	1.0
R8G8_SINT				DX10/OGL	R	G	0.0	1.0
R8G8_UINT				DX10/OGL	R	G	0.0	1.0
R16_UNORM		Yes		DX10/OGL	R	0.0	0.0	1.0
R16_SNORM				DX10/OGL	R	0.0	0.0	1.0
R16_SINT				DX10/OGL	R	0.0	0.0	1.0
R16_UINT				DX10/OGL	R	0.0	0.0	1.0
R16_FLOAT				DX10/OGL	R	0.0	0.0	1.0
A8P8_UNORM_PALETTE0					R	G	B	A
A8P8_UNORM_PALETTE1					R	G	B	A
I16_UNORM		Yes			I	I	I	I
L16_UNORM		Yes			L	L	L	1.0
A16_UNORM		Yes			0.0	0.0	0.0	A
L8A8_UNORM			Yes		L	L	L	A
I16_FLOAT		Yes			I	I	I	I
L16_FLOAT		Yes			L	L	L	1.0
A16_FLOAT		Yes			0.0	0.0	0.0	A
L8A8_UNORM_SRGB					L	L	L	A

Surface Format Name	Filtering	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A
R5G5_SNORM_B6_UNORM			Yes		R	G	B	1.0
P8A8_UNORM_PALETTE0					R	G	B	A
P8A8_UNORM_PALETTE1					R	G	B	A
A1B5G5R5_UNORM	CHV, BSW				R	G	B	A
A4B4G4R4_UNORM	CHV, BSW				R	G	B	A
R8_UNORM			Yes	DX10/OGL	R	0.0	0.0	1.0
R8_SNORM				DX10/OGL	R	0.0	0.0	1.0
R8_SINT				DX10/OGL	R	0.0	0.0	1.0
R8_UINT				DX10/OGL	R	0.0	0.0	1.0
A8_UNORM			Yes		0.0	0.0	0.0	A
I8_UNORM					I	I	I	I
L8_UNORM			Yes		L	L	L	1.0
P4A4_UNORM_PALETTE0					R	G	B	A
A4P4_UNORM_PALETTE0					R	G	B	A
P8_UNORM_PALETTE0					R	G	B	A
L8_UNORM_SRGB					L	L	L	1.0
P8_UNORM_PALETTE1					R	G	B	A
P4A4_UNORM_PALETTE1					R	G	B	A
A4P4_UNORM_PALETTE1					R	G	B	A
DXT1_RGB_SRGB					R	G	B	1.0
R1_UNORM					R	0.0	0.0	1.0
P2_UNORM_PALETTE0					R	G	B	A
P2_UNORM_PALETTE1					R	G	B	A
BC1_UNORM			Yes		R	G	B	A
BC2_UNORM			Yes		R	G	B	A
BC3_UNORM			Yes		R	G	B	A
BC4_UNORM				DX10/OGL	R	0.0	0.0	1.0
BC5_UNORM				DX10/OGL	R	G	0.0	1.0
BC1_UNORM_SRGB					R	G	B	A
BC2_UNORM_SRGB					R	G	B	A
BC3_UNORM_SRGB					R	G	B	A
MONO8					N/A	N/A	N/A	N/A

Surface Format Name	Filtering	Shadow Map	Chroma Key	Border Color Mode	R	G	B	A
DXT1_RGB					R	G	B	1.0
FXT1					R	G	B	A
R8G8B8_UNORM	CHV, BSW				R	G	B	1.0
R8G8B8_SNORM	CHV, BSW				R	G	B	1.0
BC4_SNORM				DX10/OGL	R	0.0	0.0	1.0
BC5_SNORM				DX10/OGL	R	G	0.0	1.0
R16G16B16_FLOAT					R	G	B	1.0
R16G16B16_UNORM	CHV, BSW				R	G	B	1.0
R16G16B16_SNORM	CHV, BSW				R	G	B	1.0
BC6H_SF16	CHV, BSW				R	G	B	1.0*
BC7_UNORM	CHV, BSW				R	G	B	A
BC7_UNORM_SRGB	CHV, BSW				R	G	B	A
BC6H_UF16	CHV, BSW				R	G	B	1.0*
PLANAR_420_8	CHV, BSW				Cr	Y	Cb	1.0
R8G8B8_UNORM_SRGB	CHV, BSW				R	G	B	1.0
ETC1_RGB8	CHV, BSW				R	G	B	1.0
ETC2_RGB8	CHV, BSW				R	G	B	1.0
EAC_R11	CHV, BSW				R	0.0	0.0	1.0
EAC_RG11	CHV, BSW				R	G	0.0	1.0
EAC_SIGNED_R11	CHV, BSW				R	0.0	0.0	1.0
EAC_SIGNED_RG11	CHV, BSW				R	G	0.0	1.0
ETC2_SRGB8	CHV, BSW				R	G	B	1.0
R16G16B16_UINT				DX10/OGL	R	G	B	1.0
R16G16B16_SINT				DX10/OGL	R	G	B	1.0
ETC2_RGB8_PTA	CHV, BSW				R	G	B	A
ETC2_SRGB8_PTA	CHV, BSW				R	G	B	A
ETC2_EAC_RGBA8	CHV, BSW				R	G	B	A
ETC2_EAC_SRGB8_A8	CHV, BSW				R	G	B	A
R8G8B8_UINT				DX10/OGL	R	G	B	1.0
R8G8B8_SINT				DX10/OGL	R	G	B	1.0
ASTC	CHV, BSW				R	G	B	A

Programming Note	
Context:	SURFACE_STATE/Shader channel select
It is recommended, for performance reasons, to never use any format of the type L*A*, I* or A*. Instead use R* or RG* in combination with **Shader Channel Select**.	

Programming Note	
Context:	SURFACE_STATE/Shader channel select
The BC2_NORM, BC3_UNORM, BC5_UNORM, BC5_SNORM and BC7_UNORM surface types must only be used when the **Sampler L2 Bypass Mode Disable** field in the RENDER_SURFACE_STATE is set.	

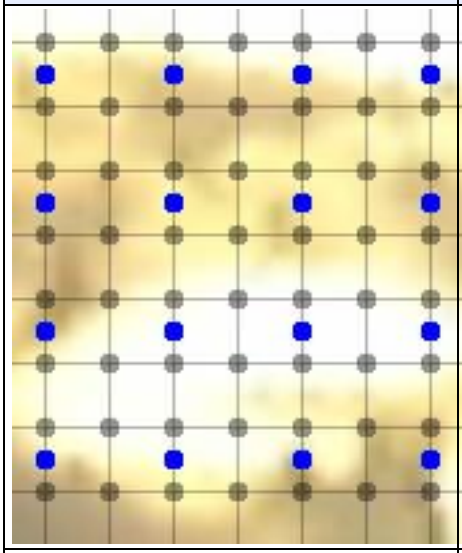
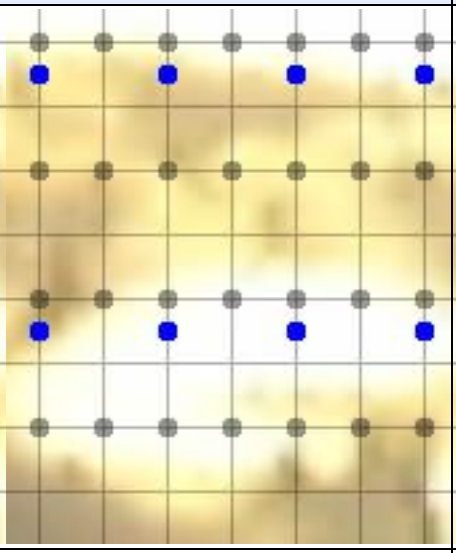
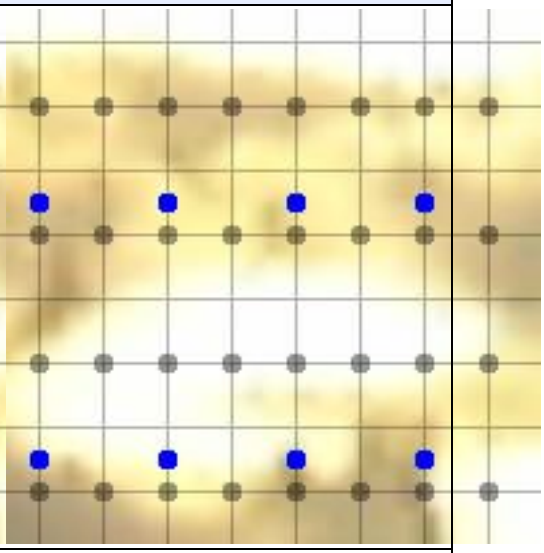
Programming Note	
Context:	NULL Surfaces and Shader Channel Select
Is SURFTYPE_NULL is selected, **Shader Channel Select Alpha** must be programmed to SCS_ZERO	

SURFACE_STATE for Deinterlace sample_8x8 and VME

This section contains media surface state definitions.

Cr(V)/Cb(U) Pixel Offset V Direction

The position of Y is brown and the position of Cr(V)/Cb(U) is blue.

Full Frame	Top Field	Bottom Field
		
V Offset 0.5	V Offset 0.25	V Offset 0.75

MEDIA_SURFACE_STATE

Programming Note	
Context:	SURFACE_STATE for Deinterlace sample_8x
The Faulting modes described in the MEMORY_OBJECT_CONTROL_STATE should be set to the same for the multi-surface Video Analytics functions like “LBP Correlation” and “Correlation Search” for both the surfaces.	

SAMPLER_STATE

SAMPLER_STATE has different formats, depending on the message type used. For CHV, BSW, the sample_8x8 and deinterlace messages use a different format of SAMPLER_STATE as detailed in the corresponding sections.

Restriction: The **Min LOD** and **Max LOD** fields need range increased from [0.0,13.0] to [0.0,14.0] and fractional bits increased from six to eight. This requires a few fields to be moved as indicated in the text.

SAMPLER_STATE

SAMPLER_BORDER_COLOR_STATE

If border color is used, all formats must be provided. Hardware will choose the appropriate format based on **Surface Format** and **Texture Border Color Mode**. The values represented by each format should be the same (other than being subject to range-based clamping and precision) to avoid unexpected behavior.

DWord	Bits	Description
0	31:24	**Border Color Alpha** Format = UNORM8
	23:16	**Border Color Blue** Format = UNORM8
	15:8	**Border Color Green** Format = UNORM8
	7:0	**Border Color Red** Format = UNORM8
1	31:0	**Border Color Red** Format = IEEE_FP
2	31:0	**Border Color Green** Format = IEEE_FP
3	31:0	**Border Color Blue** Format = IEEE_FP
4	31:0	**Border Color Alpha** Format = IEEE_FP
5	31:16	**Border Color Green** Format = FLOAT16
	15:0	**Border Color Red** Format = FLOAT16
6	31:16	**Border Color Alpha** Format = FLOAT16
	15:0	**Border Color Blue** Format = FLOAT16
7	31:16	**Border Color Green** Format = UNORM16
	15:0	**Border Color Red**

DWord	Bits	Description
		Format = UNORM16
8	31:16	**Border Color Alpha** Format = UNORM16
	15:0	**Border Color Blue** Format = UNORM16
9	31:16	**Border Color Green** Format = SNORM16
	15:0	**Border Color Red** Format = SNORM16
10	31:16	**Border Color Alpha** Format = SNORM16
	15:0	**Border Color Blue** Format = SNORM16
11	31:24	**Border Color Alpha** Format = SNORM8
	23:16	**Border Color Blue** Format = SNORM8
	15:8	**Border Color Green** Format = SNORM8
	7:0	**Border Color Red** Format = SNORM8

Border Color Programming for Integer Surface Formats

For integer formats, there are different possible cases depending on the bits per channel (bpc) and bits per texel (bpt) of the surface format.

Integer Surface Format – Different Types	Surface formats
32 bpc, 128 bpt case (4 types)	R32G32B32A32_UINT R32G32B32_UINT R32G32B32A32_SINT R32G32B32_SINT
16 bpc, 64 bpt case (5 types)	R16G16B16A16_UINT, R10G10B10A2_UINT X32_TYPELESS_G8X24_UINT R16G16B16_UINT R16G16B16A16_SINT R16G16B16_SINT
32 bpc, 64 bpt case (2 types)	R32G32_UINT

Integer Surface Format – Different Types	Surface formats
	R32G32_SINT
8 bpc, 32 bpt cases (9 types)	R8G8B8A8_UINT R8G8_UINT R8_UINT X24_TYPELESS_G8_UINT R8G8B8_UINT R8G8B8A8_SINT R8G8_SINT R8_SINT R8G8B8_SINT
16 bpc, 32 bpt cases (4 types)	R16G16_UINT R16_UINT R16G16_SINT R16_SINT
32 bpc, 32 bpt case (2 types)	R32_UINT R32_SINT

HW supports only 1 index for a given Sampler Border Color state and Sampler State. So, SW will have to program the table in **SAMPLER_BORDER_COLOR_STATE** at DWord offsets 16 to 19, as per the integer surface format type (depends on the bits per channel and bits per texel of the surface format). If any color channel is missing from the surface format, the corresponding border color should be programmed as zero; if the alpha channel is missing, the corresponding Alpha border color should be programmed as 1. Some of the representative cases are listed below:

Case 2: R32G32B32A32_SINT (32 bpc, 128 bpt, 4 channel, SINT)

Each of the values in the above table would have be to programmed as sint32 value.

Case 3: R32G32B32_UINT (32 bpc, 128 bpt, 3 channel)

R/G/B values would be programmed like in Case1. Alpha channel value at DWORDN+3 would have to be programmed as Integer 1.

Messages

The sampler receives messages from shader clients. These messages contain information to allow the sampler to perform sample operations and return results. A message consists of four components:

- Execution Mask : Indicates, for a given SIMD, which pixels are valid.
- Message Descriptor: Required information including length of the message, and the lenght of the response
- Message Header: Optional information which may be required for certain operations (e.g. Direct Write to Render Target)
- Message Payload: Specific data for each sampler operation including coordinates and other message parameters.

Programming Notes

A message header is required for GPGPU kernals in order to allow mid-thread pre-emption to allow save/restore mechanisms to work correctly.

Message Descriptor and Execution Mask

Execution Mask

SIMD16. The 16-bit execution mask forms the valid pixel signals. This determines which pixels are sampled and results returned to the GRF registers. Samples for invalid pixels are not overwritten in the GRF. However, if LOD needs to be computed for a subspan based on the message type and MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, as these are needed for the LOD computation.

SIMD8. The low 8 bits of the execution mask form the valid pixel signals. If LOD needs to be computed based on MIP filter mode and at least one pixel in the subspan being valid, the sampling engine assumes that the parameters for the upper left, upper right, and lower left pixels in the subspan are valid regardless of the execution mask, since these are needed for the LOD computation.

SIMD4x2. The low 8 bits of the execution mask is interpreted in groups of four. If any of the high 4 bits are asserted, that sample is valid. If any of the low 4 bits are asserted, that sample is valid. The **Write Channel Mask** rather than the execution mask determines which channels are written back to the GRF.

SIMD32. The execution mask is ignored, all pixels are considered valid, and all channels are returned regardless of the execution mask.

Message Descriptor

Message Descriptor - Sampling Engine

Message Header

The message header for the sampling engine is the same regardless of the message type. The messasge header is optional. If the header is not present, the behavior is as if the message was sent with all fields in the header set to zero and the write channel masks are all enabled and offsets are zero. However, if the header is not included in the message, the Sampler State Pointer will be obtained from the command stream input for the given thread. When Response length is 0 for sample_8x8 message then the data from sampler is directly written out to memory using media write message.

DWord	Bits	Description
M0.5	31:0	Reserved
M0.4	31:0	Reserved
M0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the sampler state table. This field is ignored for *ld* and *resinfo* message types. This pointer is relative to the **Dynamic State Base Address**. Format = DynamicStateOfffset[31:5] The Sampler State Pointer does not have to be defined by the Message Header (many messages do not require a message header). The Sampler State Pointer may be delivered from the Command Streamer without the need for a Message Header.
	4:0	Ignored
M0.2 spans so many rows, many for various projects, that the DWord value is repeated in each row.		
M0.2	31:22	Reserved
M0.2	21	Reserved
M0.2	20	Reserved
M0.2	19:18	**SIMD32/64 Output Format Control** Specifies the output format of SIMD32/64 messages (sample_unorm* and sample_8x8). Ignored for other message types. Refer to the writeback message formats for details on how this field affects returned data. This field is ignored for sample_8x8 messages if the Function is not AVS and MinMaxFilter. For MinMaxFilter only 16 bit Full and 8 bit Full modes are supported. 0: 16 bit Full 1: 16 bit Chrominance Downsampled 2: 8 bit Full 3: 8 bit Chrominance Downsampled

DWord	Bits	Description
		This feature should be programmed to 0h because non-0 values may cause data corruption in returned values.
M0.2	17:16	**Gather4 Source Channel Select:** Selects the source channel to be sampled in the gather4* messages. Ignored for other message types. 0: Red channel 1: Green channel 2: Blue channel 3: Alpha channel For gather4*_c messages, this field must be set to 0 (Red channel).
M0.2	15	**Alpha Write Channel Mask:** Enables the alpha channel to be written back to the originating thread. 0: Alpha channel is written back. 1: Alpha channel is not written back. **Programming Note** **Project:** CHV, BSW **Context:** 3D Sampler Messages • A message with all four channels masked is not allowed. • This field is ignored for the deinterlace message. • This field must be set to zero for sample_8x8 in VSA mode. • Alpha/Blue/Green/Red channels masked must set to 0 (no mask is supported). • CHV, BSW: For Sample_8x8 messages, Alpha/Blue/Red channels should be always masked (set to 1) and only Green channel is enabled (set to 0). • This field must be set to zero for all gather4* messages. • CHV, BSW: This field must be set to zero for all sample_min/sample_max instructions
M0.2	15	**Alpha Write Channel Mask:** Enables the alpha channel to be written back to the originating thread.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>0: Alpha channel is written back.
1: Alpha channel is not written back.
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>3D Sampler Messages</td></tr><tr><td colspan="2">• A message with all four channels masked is not allowed.
• This field is ignored for the deinterlace message.
• This field must be set to zero for sample_8x8 in VSA mode.
• CHV, BSW: For Sample_8x8 messages, Alpha/Blue/Red channels should be always masked (set to 1) and only Green channel is enabled (set to 0).
• This field must be set to zero for all gather4* messages.
• CHV, BSW: This field must be set to zero for all sample_min/sample_max instructions</td></tr></table></td></tr>
<tr><td>M0.2</td><td>14</td><td>Blue Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>13</td><td>Green Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>12</td><td>Red Write Channel Mask: See Alpha Write Channel Mask.</td></tr>
<tr><td>M0.2</td><td>11.8</td><td>U Offset: The u offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>3D Sampler Messages</td></tr><tr><td colspan="2">• This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
• This field is ignored if the offu parameter is included in the gather4* messages.</td></tr></table><table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>Non-Normalized Floating-Point Coordinates</td></tr><tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr></table></td></tr>
</table>

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>M0.2</td><td>11:8</td><td>U Offset: The u offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.</td></tr>
<tr><td>M0.2</td><td>7:4</td><td>V Offset: The v offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>3DSampler Messages</td></tr>
<tr><td colspan="2">• This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
• This field is ignored if the offu parameter is included in the gather4* messages.</td></tr>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>Non-Normalized Floating-Point Coordinates</td></tr>
<tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr>
</table></td></tr>
<tr><td>M0.2</td><td>3:0</td><td>R Offset: The r offset from the _aoffimmi modifier on the sample or ld instruction in DX10. Must be zero if the Surface Type is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2’s complement.
<table>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>3D Sampler Messages</td></tr>
<tr><td colspan="2">This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.</td></tr>
<tr><th colspan="2">Programming Note</th></tr>
<tr><td>Context:</td><td>Non-Normalized Floating-Point Coordinates</td></tr>
<tr><td colspan="2">Texel offsets can only be applied to messages with floating-point normalized coordinates or integer non-normalized coordinates.</td></tr>
</table></td></tr>
<tr><td>M0.1</td><td>31:0</td><td>Reserved</td></tr>
<tr><td>M0.0</td><td>31:0</td><td>Reserved</td></tr>
</table>

Parameter Types

sample*, LOD, and gather4 messages

For all of the sample*, LOD, and gather4 message types, all parameters are 32-bit floating point, except the 'mcs', 'offu', and 'offv' parameters. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Normalized values range from [0,1] across the surface, with values outside the surface behaving as specified by the **Address Control Mode** in that dimension. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension, with values outside the surface being clamped to the surface.

Surface Type	u	v	r	ai
SURFTYPE1D	normalized 'x' coordinate	unnormalized array index	ignored	ignored
SURFTYPE_2D	normalized 'x' coordinate	normalized 'y' coordinate	unnormalized array index	ignored
SURFTYPE_3D	normalized 'x' coordinate	normalized 'y' coordinate	normalized 'z' coordinate	ignored
SURFTYPE_CUBE	normalized 'x' coordinate	normalized 'y' coordinate	normalized 'z' coordinate	unnormalized array index

mcs parameter

The 'mcs' parameter delivers the multisample control data. The format of this parameter is always a 32-bit unsigned integer. Refer to the section titled "Multisampled Surface Behavior" for details on this parameter.

Ld* messages

For the ld message types, all parameters are 32-bit unsigned integers, except the 'mcs' parameter. Usage of the u, v, and r parameters is as follows based on **Surface Type**. Unnormalized values range from [0,n-1] across the surface, where n is the size of the surface in that dimension. Input of any value outside of the range returns zero.

Surface Type	u	v	r
SURFTYPE1D	unnormalized 'x' coordinate	unnormalized array index	ignored
SURFTYPE_2D	unnormalized 'x' coordinate	unnormalized 'y' coordinate	unnormalized array index
SURFTYPE_3D	unnormalized 'x' coordinate	unnormalized 'y' coordinate	unnormalized 'z' coordinate
SURFTYPE_BUFFER	unnormalized 'x' coordinate	ignored	ignored

SIMD Payloads

This section contains the SIMD payload definitions.

SIMD16 Payload [CHV, BSW]

The payload of a SIMD16 message provides addresses for the sampling engine to process 16 entities (examples of an entity are vertex and pixel). The number of parameters required to sample the surface depends on the state that the sampler/surface is in. Each parameter takes two message registers, with 8 entities, each a 32-bit floating point value, being placed in each register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this. For example, a 2D map using "sample_b" needs only u, v, and bias, but must send the r parameter as well.

DWord	Bits	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2.7	31:0	**Subspan 3, Pixel 3 (lower right) Parameter 0**
M2.6	31:0	**Subspan 3, Pixel 2 (lower left) Parameter 0**
M2.5	31:0	**Subspan 3, Pixel 1 (upper right) Parameter 0**
M2.4	31:0	**Subspan 3, Pixel 0 (upper left) Parameter 0**
M2.3	31:0	**Subspan 2, Pixel 3 (lower right) Parameter 0**
M2.2	31:0	**Subspan 2, Pixel 2 (lower left) Parameter 0**
M2.1	31:0	**Subspan 2, Pixel 1 (upper right) Parameter 0**
M2.0	31:0	**Subspan 2, Pixel 0 (upper left) Parameter 0**
M3 – Mn		Repeat packets 1 and 2 to cover all required parameters.

SIMD8 Payload [CHV, BSW]

This message is intended to be used in a SIMD8 thread, or in pairs from a SIMD16 thread. Each message contains sample requests for just 8 pixels.

DWord	Bits	Description
M1.7	31:0	**Subspan 1, Pixel 3 (lower right) Parameter 0** Specifies the value of the pixel's parameter 0. The actual parameter that maps to parameter 0 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Subspan 1, Pixel 2 (lower left) Parameter 0**
M1.5	31:0	**Subspan 1, Pixel 1 (upper right) Parameter 0**
M1.4	31:0	**Subspan 1, Pixel 0 (upper left) Parameter 0**
M1.3	31:0	**Subspan 0, Pixel 3 (lower right) Parameter 0**
M1.2	31:0	**Subspan 0, Pixel 2 (lower left) Parameter 0**
M1.1	31:0	**Subspan 0, Pixel 1 (upper right) Parameter 0**
M1.0	31:0	**Subspan 0, Pixel 0 (upper left) Parameter 0**
M2 – Mn		Repeat packet 1 to cover all required parameters.

SIMD4x2 Payload [CHV, BSW]

DWord	Bits	Description
M1.7	31:0	**Sample 1 Parameter 3** Specifies the value of the pixel's parameter 3. The actual parameter that maps to parameter 3 is given in the table in the Payload Parameter Definition section. Format = IEEE Float for all sample* message types, U32 for ld and resinfo message types.
M1.6	31:0	**Sample 1 Parameter 2**
M1.5	31:0	**Sample 1 Parameter 1**
M1.4	31:0	**Sample 1 Parameter 0**
M1.3	31:0	**Sample 0 Parameter 3**
M1.2	31:0	**Sample 0 Parameter 2**
M1.1	31:0	**Sample 0 Parameter 1**
M1.0	31:0	**Sample 0 Parameter 0**
M2		Parameters 4-7 if present
M3		Parameters 8-11 if present

Writeback Message

Corresponding to the four input message definitions are four writeback messages. Each input message generates a corresponding writeback message of the same type .

SIMD16

Return Format = 32-bit

A SIMD16 writeback message consists of up to 8 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3. The pixels written within each destination register is determined by the execution mask on the "send" instruction.

DWord	Bit	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1.7	31:0	**Subspan 3, Pixel 3 (lower right) Red**
W1.6	31:0	**Subspan 3, Pixel 2 (lower left) Red**
W1.5	31:0	**Subspan 3, Pixel 1 (upper right) Red**
W1.4	31:0	**Supspan 3, Pixel 0 (upper left) Red**
W1.3	31:0	**Subspan 2, Pixel 3 (lower right) Red**
W1.2	31:0	**Subspan 2, Pixel 2 (lower left) Red**
W1.1	31:0	**Subspan 2, Pixel 1 (upper right) Red**
W1.0	31:0	**Supspan 2, Pixel 0 (upper left) Red**
W2		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W3		**Subspans 3 and 2 of Green:** See W1 definition for pixel locations
W4		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W5		**Subspans 3 and 2 of Blue:** See W1 definition for pixel locations
W6		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations

DWord	Bit	Description
W7		**Subspans 3 and 2 of Alpha:** See W1 definition for pixel locations
W8.7:1		Reserved (not written): W8 is only delivered when **Pixel Fault Mask Enable** is enabled.
W8.0	15:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

Return Format = 16-bit

[CHV, BSW]: A SIMD16 writeback message with **Return Format** of 16-bit consists of up to 4 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the "send" instruction.

DWord	Bit	Description
W0.7	31:16	**Subspan 3, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Half Float, S15 signed 2's comp integer, or U16 unsigned integer.
	15:0	**Subspan 3, Pixel 2 (lower left) Red**
W0.6	31:16	**Subspan 3, Pixel 1 (upper right) Red**
	15:0	**Supspan 3, Pixel 0 (upper left) Red**
W0.5	31:16	**Subspan 2, Pixel 3 (lower right) Red**
	15:0	**Subspan 2, Pixel 2 (lower left) Red**
W0.4	31:16	**Subspan 2, Pixel 1 (upper right) Red**
	15:0	**Supspan 2, Pixel 0 (upper left) Red**
W0.3	31:16	**Subspan 1, Pixel 3 (lower right) Red**
	15:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.2	31:16	**Subspan 1, Pixel 1 (upper right) Red**
	15:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.1	31:16	**Subspan 0, Pixel 3 (lower right) Red**
	15:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.0	31:16	**Subspan 0, Pixel 1 (upper right) Red**
	15:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Green:** See W0 definition for pixel locations
W2		**Blue:** See W0 definition for pixel locations
W3		**Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written): W4 is only delivered when **Pixel Fault Mask Enable** is enabled.
W4.0	31:16	Reserved: always written as 0xffff
W4.0	15:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null

DWord	Bit	Description
		page was source for at least one texel.

SIMD8/SIMD8D

Return Format = 32-bit

[CHV, BSW]: A SIMD8* writeback message consists of up to 4 destination registers (5 in the case of sample+killpix). Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the *send* instruction.

For the sample+killpix message types, an additional register (W4) is included after the last channel register.

DWord	Bits	Description
W0.7	31:0	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Float, S31 signed 2's comp integer, or U32 unsigned integer.
W0.6	31:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.5	31:0	**Subspan 1, Pixel 1 (upper right) Red**
W0.4	31:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.3	31:0	**Subspan 0, Pixel 3 (lower right) Red**
W0.2	31:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.1	31:0	**Subspan 0, Pixel 1 (upper right) Red**
W0.0	31:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W2		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W3		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written) **:** This W4 is only delivered for the sample+killpix message type
W4.0	31:16	**Dispatch Pixel Mask:** This field is always 0xffff to allow dword-based ANDing with the R0 header in the pixel shader thread.
W4.0	15:0	**Active Pixel Mask:** This field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode. Since the SIMD8 message applies to only 8 pixels, only the low 8 bits within this field are used. The high 8 bits are always set to 1.
W4.7:1		Reserved (not written): This W4 is only delivered when **Pixel Fault Mask Enable** is enabled.
W4.0	31:24	Reserved: always written as 0xff
W4.0	23:16	Reseved: always written as 0xff
W4.0	15:8	Reserved: always written as 0xff

DWord	Bits	Description
W4.0	7:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

Return Format = 16-bit

SIMD8* writeback message with **Return Format** of 16-bit consists of up to 4 destination registers). Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The pixels written within each destination register is determined by the execution mask on the send instruction.

DWord	Bits	Description
W0.7:4		Reserved
W0.3	31:16	**Subspan 1, Pixel 3 (lower right) Red:** Specifies the value of the pixel's red channel. Format = IEEE Half Float, S15 signed 2's comp integer, or U16 unsigned integer.
	15:0	**Subspan 1, Pixel 2 (lower left) Red**
W0.2	31:16	**Subspan 1, Pixel 1 (upper right) Red**
	15:0	**Supspan 1, Pixel 0 (upper left) Red**
W0.1	31:16	**Subspan 0, Pixel 3 (lower right) Red**
	15:0	**Subspan 0, Pixel 2 (lower left) Red**
W0.0	31:16	**Subspan 0, Pixel 1 (upper right) Red**
	15:0	**Supspan 0, Pixel 0 (upper left) Red**
W1		**Subspans 1 and 0 of Green:** See W0 definition for pixel locations
W2		**Subspans 1 and 0 of Blue:** See W0 definition for pixel locations
W3		**Subspans 1 and 0 of Alpha:** See W0 definition for pixel locations
W4.7:1		Reserved (not written): This W4 is only delivered when **Pixel Fault Mask Enable** is enabled.
W4.0	31:24	Reserved: always written as 0xffff
W4.0	23:16	Reserved: always written as 0xff
W4.0	15:8	Reserved: always written as 0xff
W4.0	7:0	**Pixel Null Mask:** This field has the bit for all pixels set to 1 except those pixels in which a null page was source for at least one texel.

SIMD4x2

A SIMD4x2 writeback message always consists of a single message register containing all four channels of each of the two “pixels” (called “samples” here, as they are not really pixels) of data. The write channel mask bits as well as the execution mask on the “send” instruction are used to determine which of the channels in the destination register are overwritten. If any of the four execution mask bits for a sample is asserted, that sample is considered to be active. The active channels in the write channel mask will be written in the destination register for that sample. If the sample is inactive (all four execution mask bits deasserted), none of the channels for that sample will be written in the destination register.

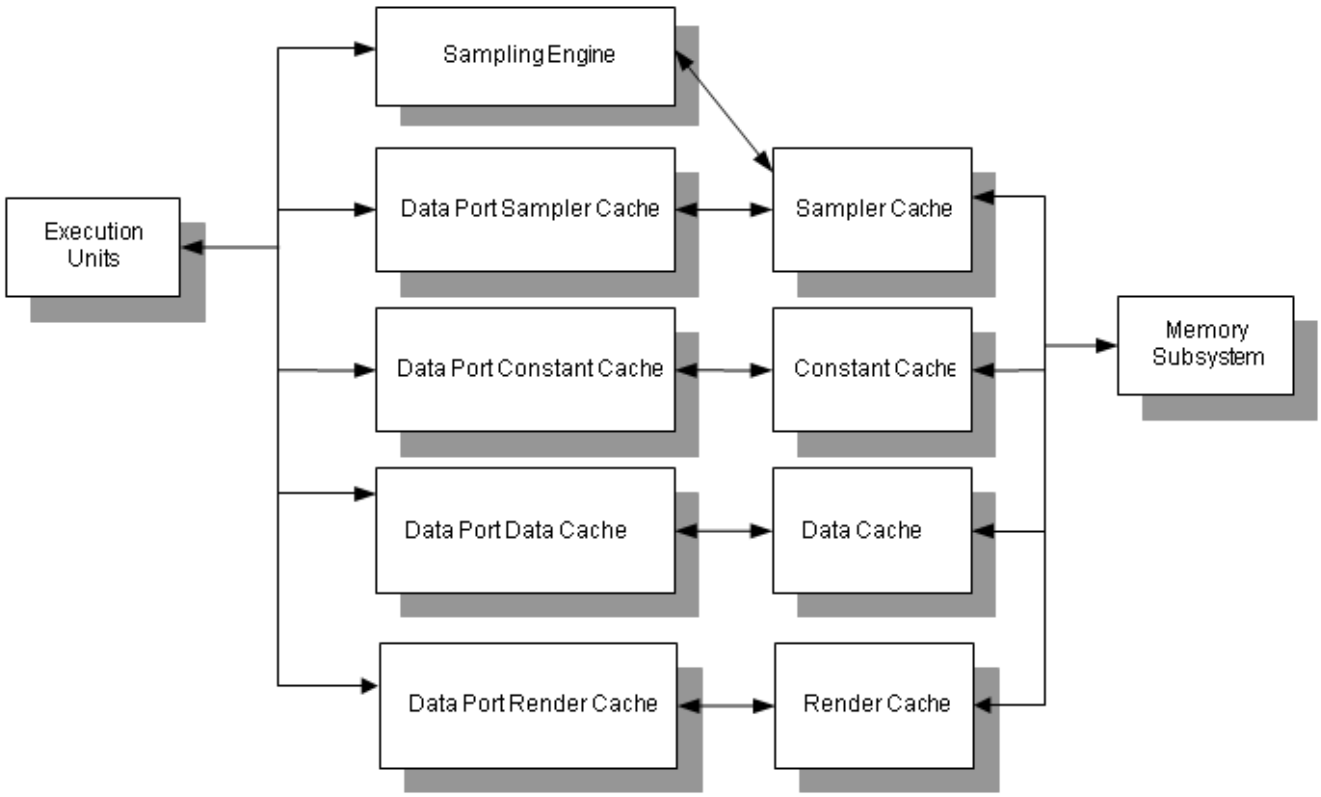
DWord	Bit	Description
W0.7	31:0	**Sample 1 Alpha:** Specifies the value of the pixel’s alpha channel. Format = IEEE Float, S31 signed 2’s comp integer, or U32 unsigned integer.
W0.6	31:0	**Sample 1 Blue**
W0.5	31:0	**Sample 1 Green**
W0.4	31:0	**Sample 1 Red**
W0.3	31:0	**Sample 0 Alpha**
W0.2	31:0	**Sample 0 Blue**
W0.1	31:0	**Sample 0 Green**
W0.0	31:0	**Sample 0 Red**
W1.7:1		Reserved (not written) : W1 is only delivered when **Pixel Fault Mask Enable** is enabled.
W1.0	31:2	Reserved: always written as 0x3fffffff
	1:0	**Pixel Null Mask:** This field has the bit for all samples set to 1 except those pixels in which a null page was source for at least one texel.

Shared Functions – Data Port

The Data Port provides all memory accesses for the Gen subsystem other than those provided by the sampling engine. These include render target writes, constant buffer reads, scratch space reads/writes, and media surface accesses.

CHV, BSW adds the Data Port Data Cache and the Data Cache.

The diagram below shows the four parts of the Data Port (Sampler Cache, Constant Cache, Data Cache, and Render Cache) and how they connect with the caches and memory subsystem. The execution units and sampling engine are shown for clarity.



The kernel programs running in the execution units communicate with the data port via messages, the same as for the other shared function units. The four data ports are considered to be separate shared functions, each with its own shared function identifier.

Data Cache

The data cache is a read/write cache that is coherent across the physical instances of this cache. It is intended to be used for the following surfaces:

- constant buffers
- destination surfaces for media applications
- intermediate working surfaces for media applications
- scratch space buffers
- general read/write access of surfaces
- atomic operations
- shared memory for GPGPU thread groups

The data cache can be accessed via the *Data Cache Data Port* shared function, and via the load and store EU messages. Ordering from a single thread is maintained when accessing the data cache using only one of these mechanisms, but is not maintained when using both of these mechanisms from the same thread. In these instances, software must ensure ordering by using write commits and/or waiting for read data to be returned.

Data Cache Coherency

Data cache memory is backed by system memory. The read/write data cache is always coherent between GPU thread accesses, but might not be coherent with CPU memory accesses.

Coherency Type
Two types of coherency are supported for data cache memory: GPU Coherent and IA Coherent. GPU coherent accesses are kept coherent with GPU threads but not kept coherent with CPU (LLC) accesses. IA Coherent accesses are kept coherent with both GPU threads and with CPU accesses. Two GPU accesses made to the same address but with different coherency types have different cache entries and are not kept coherent with each other. If a buffer will be accessed with different coherency types, software is responsible for flushing the GPU cache to system memory, using either a Memory Fence message or a pipeline flush command. The surface state field **Coherency Type** specifes the data cache coherency used with that surface. Stateless accesses specify their coherency with BTI=255 or BTI=253.

Sampler Cache

The sampler cache is a read-only cache that supports both linear and tiled memory. In addition to being used by the sampling engine (via the sampling engine messages), the sampler cache is intended to be used for source surfaces in media applications via the data port. The same application may use the sampler cache via the sampling engine and data port without flushing the pipeline between accesses.

Surfaces

The data elements accessed by the data port are called “surfaces”. There are two models used by the data port to access these surfaces: surface state model and stateless model.

Surface State Model

The data port uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine. The surface state model is used when a **Binding Table Index** (specified in the message descriptor) of less than 255 is specified. In this model, the **Binding Table Index** is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE. SURFACE_STATE contains the parameters defining the surface to be accessed, including its location, format, and size.

This model is intended to be used for constant buffers, render target surfaces, and media surfaces.

Stateless Model

The stateless model is used when a **Binding Table Index** (specified in the message descriptor) of 255 is specified.

This model is primarily intended to be used for scratch space buffers.

In this model, the binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER
- Surface Format = R32G32B32A32_FLOAT
- Vertical Line Stride = 0
- Surface Base Address = General State Base Address + Immediate Base Address
- Surface Pitch = 16 bytes
- Utilize Fence = false
- Tiled = false

Buffer Size Checking

Buffer Size Checking
Buffer Size = checked only against General State Buffer Size When General State Buffer Size is zero, any stateless access is out-of-bounds.

Shared Local Memory (SLM)

The shared local memory (SLM) is a high bandwidth memory that is not backed up by system memory. It is enabled by configuring the L3 cache to use a portion of its space for the SLM. One SLM is present in each half slice, and its contents are shared between all of the active threads in that half slice. Its contents are uninitialized after creation, and its contents disappear when deallocated.

The SLM is accessed when a **Binding Table Index** (specified in the message descriptor) of 254 is specified. The binding table is not accessed, and the parameters that define the surface state are overloaded as follows:

- Surface Type = SURFTYPE_BUFFER
- Surface Format = RAW
- Surface Base Address = points to the start of the internal SLM (no memory address is applicable)
- Surface Pitch = 1 byte

Due to the predefined surface state attributes for the SLM, only a subset of the data port messages can be used. This includes the Byte Scattered Read/Write, Untyped Surface Read/Write, and Untyped Atomic Operation messages. In addition, only the data cache data port is supported; the other data ports treat Binding Table Index 254 as a normal surface state access.

Programming Note	
Context:	Shared Local Memory (SLM)
Accesses to SLM don't have any bounds checking. Addresses beyond the size (64KB) of the SLM wrap around.	

Write Commit

For write messages, an optional write commit writeback message can be requested via the Send Write Commit Message bit in the message descriptor. This bit causes a return message to the thread indicating when the write has been committed to the in-order cache pipeline and it is safe to issue another access to the same data with the assurance that it will happen after the first write. A read issued after the write commit ensures that the read will get the newly written data, and another write issued after the write commit will be the last to modify the data. "Committed" does not guarantee that the data has been actually written to the memory subsystem, but only that the write has been scheduled and cannot be passed by another read or write issued subsequently.

If **Send Write Commit Message** is used on a Flush Render Cache message, the write commit is sent only when the render cache has completed its flush to memory. A read issued to another cache after the write commit is received will be guaranteed to retrieve the “new” data that was written before the Flush Render Cache message was issued.

The write commit does not modify the destination register, but merely clears the dependency associated with the destination register. Thus, a simple “mov” instruction using the register as a source is sufficient to wait for the write commit to occur. The following code sequence indicates this:

```
send r12 m1 DPWRITE   ; Issue write to render cache.
mov m1 r3             ; Assemble read message.
mov r12 r12           ; Block on write commit.
send r13 m1 DPREAD    ; Read same location as write.
```

ReadWrite Ordering

Reads and writes issued from the same thread *are* guaranteed to be processed in the same order as issued. Software mechanisms must still ensure any needed ordering of accesses issued from different threads.

Accessing Buffers

There are four data port messages used to access buffers. Three of these are used for both constant buffers and scratch space buffers, the fourth is used by the geometry shader kernel to write to streamed vertex buffers. All of these messages support only buffers, and can use the surface state model as well as the stateless model.

The following table indicates the intended applications of each of the buffer messages.

Message	Applications
OWord Block Read/Write	• constant buffer reads of a single constant or multiple contiguous constants • scratch space reads/writes where the index for each pixel/vertex is the same • block constant reads, scratch memory reads/writes for media
OWord Dual Block Read/Write	• SIMD4x2 constant buffer reads where the indices of each vertex/pixel are different (if there are two indices and they are the same, hardware will optimize the cache accesses and do only one cache access) • SIMD4x2 scratch space reads/writes where the indices are different.
DWord Scattered Read/Write	• SIMD8/16 constant buffer reads where the indices of each pixel are different (read one channel per message) • SIMD8/16 scratch space reads/writes where the indices are different (read/write one channel per message) • general purpose DWord scatter/gathering, used by media
Streamed Vertex Buffer Write	• geometry shader streaming vertex data out

These messages generally ignore the surface format field of the state and perform no format conversion. The exception is the Streamed Vertex Buffer Write, which uses the surface format field to determine only how many channels are to be written. The data contained in each channel is still not converted in any way.

Accessing Media Surfaces

The Media Block Read/Write message is intended to be used to access 2D media surfaces. The message specifies an X/Y coordinate into the 2D surface as input. Since this message only supports 2D surfaces, the stateless model cannot be used with this message.

Boundary Behavior

The table below summarizes the behavior of the **Media Boundary Pixel Mode** field (SURFACE_STATE) in combination with the **Vertical Line Stride** and **Vertical Line Stride Offset** fields (both of which are subject to being overridden by the Data Port message descriptor fields). The Behavior column illustrates behavior for a surface with four rows numbered 0 to 3. The bold indicators are off-surface behavior and the non-bold indicators are on-surface behavior. Input row addresses range from -3 to +7 going left to right.

Media Boundary Pixel Mode	Vertical Line Stride	Vertical Line Stride Offset	Usage Model	Behavior
0	0	X	normal frame	000001233333
0	1	0	normal field even	000002222222
0	1	1	normal field odd	111113333333
2	0	X	frame / progressive	000001233333
2	1	0	field even / progressive	000002333333
2	1	1	field odd / progressive	000013333333
3	0	X	frame / interlaced	010101232323
3	1	0	field even / interlaced	000002222222
3	1	1	field odd / interlaced	111113333333

State

This topic discusses applicable State options for the Data Port.

BINDING_TABLE_STATE

The data port uses the binding table to retrieve surface state. Refer to *State* in the Sampling Engine section for the definition of this state.

SURFACE_STATE

The data port uses the surface state for constant buffers, render targets, and media surfaces.

COLOR_PROCESSING_STATE

The following state structures contain different states used by the color processing function.

COLOR_PROCESSING_STATE - STD/STE State

COLOR_PROCESSING_STATE - ACE State

COLOR_PROCESSING_STATE - TCC State

COLOR_PROCESSING_STATE - PROCAMP State

COLOR_PROCESSING_STATE - CSC State

Messages

This section of the BSpec discusses messages applicable to the Data Port.

Global Definitions

For data port messages, part of the message descriptor is used to determine the message type. This field is documented here. The remainder of the message descriptor is defined differently depending on the message type, and is documented in the section for the corresponding message.

The Data Port is actually separate targets, **Data Port**,**Sampler Cache**, **Data Port Constant Cache**, and **Data Port Render Cache**, each with its own target unit ID. Each target has its own set of message type encodings as shown below.

Programming Note	
Context:	Messages

Data port messages may not have the **End of Thread** bit set in the message descriptor other than the following exceptions:

- The Render Target Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in non-contiguous dispatch mode.
- The Render Target UNORM Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in contiguous dispatch mode.
- The Media Block Write message may have **End of Thread** set for pixel shader threads dispatched by the windower in contiguous dispatch mode.

Data Port Messages

Most of the messages have an existing definition that is not expected to change. There are several new messages that are documented here.

Data Cache Data Port Message Summary

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
OWord Block Read	yes	no	yes	global	1
OWord Block Write	yes	no	yes	global	1
Unaligned OWord Block Read	yes	no	yes	global	1
OWord Dual Block Read	no for stated yes for stateless	no	yes	global + offset	2
OWord Dual Block Write	no for stated yes for stateless	no	yes	global + offset	2

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
A64 QWord Scattered Read	no (forbidden)	no	yes (only)	offset	8
A64 QWord Scattered Write	no (forbidden)	no	yes (only)	offset	8
A64 DWord Scattered Read	no (forbidden)	no	yes (only)	offset	8
A64 DWord Scattered Write	no (forbidden)	no	yes (only)	offset	8
A64 Byte Scattered Read	no (forbidden)	no	yes (only)	offset	8
A64 Byte Scattered Write	no (forbidden)	no	yes (only)	offset	8
A64 Untyped Surface Read	no (forbidden)	no	yes (only)	offset	2, 8
A64 Untyped Surface Write	no (forbidden)	no	yes (only)	offset	2, 8
A64 Untyped Atomic Operation	no (forbidden)	no	yes (only)	offset	8
A64 Untyped Atomic Operation SIMD4x2	no (forbidden)	no	yes (only)	offset	2
A64 Block Read/Write	yes	no	yes (only)	offset	1
DWord Scattered Read	no for stated yes for stateless	no	yes	global + offset	8, 16
DWord Scattered Write	no for stated yes for stateless	no	yes	global + offset	8, 16
Byte Scattered Read	no for stated yes for stateless	yes	yes	global + offset	8, 16
Byte Scattered Write	no for stated yes for stateless	yes	yes	global + offset	8, 16
Untyped Surface Read	no for stated yes for stateless	yes	yes (1D only)	1D or 2D	2, 8, 16
Untyped Surface Write	no for stated yes for stateless	yes	yes (1D only)	1D or 2D	2, 8, 16
Untyped Atomic Operation	no for stated	yes	yes (1D only)	1D or 2D	8, 16

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
	yes for stateless				
Untyped Atomic Operation SIMD4x2	no for stated yes for stateless	yes	yes (1D only)	1D or 2D	2
Atomic Counter Operation	no[2] Required for inc, dec, predec	no	no	implied	8
Atomic Counter Operation SIMD4x2	no[2] Required for inc, dec, predec	no	no	implied	2
Scratch Block Read	yes	no	yes (only)	Imm_Buf + offset	
Scratch Block Write	yes	no	yes (only)	Imm_Buf + offset	
Memory Fence	yes	N/A	N/A	N/A	N/A
Typed Surface Read	yes	no	no	1D, 2D, 3D, 4D	2, 8
Typed Surface Write	yes	no	no	1D, 2D, 3D, 4D	2, 8
Typed Atomic Operation	yes	no	no	1D, 2D, 3D, 4D	8
Typed Atomic Operation SIMD4x2	yes	no	no	1D, 2D, 3D, 4D	2, 8
Media Block Read	yes	no	no	2D	1
Media Block Write	yes	no	no	2D	1
Transpose Read	yes	no	no	2D	1

Table Notes

"global" is the **Global Offset** in the message header (if header is not present, Global Offset is zero).

"imm_buf" is the Immediate Buffer Base Address provided in message header register M0.5.

"offset" is in the message payload, and is per-slot.

"handle" is the handle address in the message header.

"URBoffset" is the **Global Offset** field in the URB message descriptor.

"1D" and "2D" are the address payload.

Render Cache Data Port Message Summary

Message Type	Header Required	Address Modes	Vector Width
Render Target Write	no	2D + RTAI	8, 16

Message Descriptor

This section describes the Data Port message descriptors for various projects.

Message Descriptor

SAMPLER CACHE DATA PORT		RENDER CACHE DATA PORT	
Bit	**Description**	**Bit**	**Description**
19	**Header Present.** If set, indicates that the message includes the header. **Note:** For the Sampler Cache Data Port, the header must be present for the following message types: Unaligned OWord Block Read Media block read Format = Enable		
18	Ignored	18	Ignored
17:14	Message Type 0000: Read Surface Info 0001: Unaligned OWord Block Read 0100: Media Block Read All other encodings are reserved.	17:14	Message Type 1100: Render Target Write 1101: Render Target Read All other encodings are reserved.
13:8	**Message Specific Control.** Refer to the specific message section for the definition of these bits.		
7:0	Binding Table Index. Specifies the index into the binding table for the specified surface. Format = U8 Range = [0,255]		

CONSTANT CACHE DATA PORT		DATA CACHE DATA PORT0		DATA CACHE DATA PORT1	
Bit	**Description**	**Bit**	**Description**	**Bit**	**Description**
19	Header Present. If set, indicates that the message includes the header. **Programming Note:** For the Data Cache Data Port*, the header must be present for the following message types: • OWord Block Read/Write • Unaligned OWord Block Read • Memory Fence • Scratch read/write • Typed read/write/atomics • Media block read/write				

CONSTANT CACHE DATA PORT		DATA CACHE DATA PORT0		DATA CACHE DATA PORT1	
Bit	**Description**	**Bit**	**Description**	**Bit**	**Description**
	• Transpose Read • A64 Block Read/Write For the Constant Cache Data Port, the header must be present for the following message types: • OWord Block Read/Write • Unaligned OWord Block Read • Format = Enable				
18	Ignored	18	**Category** 1: Scratch Block Read/Write messages 0: Legacy DAP-DC messages	18:14	**Message Type** 00000: Transpose Read 00001: Untyped Surface Read 00010: Untyped Atomic Operation 00011: Untyped Atomic Operation SIMD4x2 00100: Media Block Read 00101: Typed Surface Read 00110: Typed Atomic Operation 00111: Typed Atomic Operation SIMD4x2 01000: Untyped Atomic Float Add 01001: Untyped Surface Write 01010: Media Block Write 01011: Atomic Counter Operation 01100: Atomic Counter Operation SIMD4x2 01101: Typed Surface Write 10000: A64
17:14	**Message Type** 0000: OWord Block Read 0001: Unaligned OWord Block Read 0010: OWord Dual Block Read 0011: DWord Scattered Read All other encodings are reserved.	17:14	Category=0 (legacy dataport) Message Type 0000: OWord Block Read 0001: Unaligned OWord Block Read 0010: OWord Dual Block Read 0011: DWord Scattered Read 0100: Byte Scattered Read 0111: Memory Fence 1000: OWord Block Write 1010: OWord Dual Block Write 1011: DWord Scattered Write 1100: Byte Scattered Write All other encodings are reserved. Category=1 (scratch) [17]: 0=Read; 1=write [16]: Type;		

CONSTANT CACHE DATA PORT		DATA CACHE DATA PORT0		DATA CACHE DATA PORT1	
Bit	**Description**	**Bit**	**Description**	**Bit**	**Description**
			0=Oword, 1= Dword [15]: Invalidate after read; [14]: <Reserved, mbz> [13:12]: Block Size 11: 8 registers 10: 4 registers 01: 2 registers 00: 1 register [11:0]: Addr offset (Hword based)		Scattered Read 10001: A64 Untyped Surface Read 10010: A64 Untyped Atomic Operation 10011: A64 Untyped Atomic Operation SIMD4x2 10100: A64 Block Read 10101: A64 Block Write 11000: A64 Untyped Atomic Float Add 11001: A64 Untyped Surface Write 11010: A64 Scattered Write All other encodings are reserved.
13:8	**Message Specific Control.** Refer to the specific message section for the definition of these bits.				
7:0	Binding Table Index. Specifies the index into the binding table for the specified surface. For the data cache data port, two binding table indexes are used to select special surfaces: 254: A binding table index of 254 indicates that the shared local memory (SLM) is to be used. The SLM is only supported with the Byte Scattered Read/Write, Untyped Surface Read/Write, and Untyped Atomic Operation messages. Refer to the *Shared Local Memory* section earlier in this chapter for further details on its behavior. 255: A binding table index of 255 indicates that a stateless model is to be used. Refer to the *Stateless Model* section for details. 253: An alias for Stateless [CHV, BSW] Format = U8 Range = [0,255]				

[CHV, BSW] SFID_DP_DC1 is an extension of SFID_DP_DC0 to allow for more message types. They act as a single logical entity

The stateless aliases provide a means of SW controlling the coherency properties of an access. The property is ensured for that access only. Typically, SW will use the same coherency type for all access to the same address. Proper fencing is required to ensure that reads and writes are visible. L3UC forces the addressed cache lines out of L3 and the cycles are directly conducted to LLC. This provides a capability for ensuring coherency on a particular location without having to fence all the other cycles.

Binding table index	Coherency	Explanation
255	IA Coherent	Coherent within Gen and coherent within the entire IA cache memory hierarchy.
253	Non-Coherent	Coherent within Gen, same cache type.

Programming Note	
Context:	Message Descriptor
The constant, sampler and render cache are always non-coherent.	

Message Header

This header applies to the following data port messages:

Data Port Message
OWord Block Read/Write
Unaligned OWord Block Read
OWord Dual Block Read/Write
DWord Scattered Read/Write
Byte Scattered Read/Write
Scratch Block Read/Write

The header definitions for the other data port messages is in the section for each message.

DWord	Bits	Description
M0.5	31:10	**Immediate Buffer Base Address.** Specifies the surface base address for messages in which the Binding Table Index is 255 (stateless model), else this field is ignored. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Ignored
	7:0	**Dispatch ID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:4	Ignored
	3:0	**Per Thread Scratch Space** Specifies the amount of scratch space allowed to be used by this thread for messages in which the Binding Table Index is 255 (stateless model), else this field is ignored. Programming Notes: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages. The data port uses this to

DWord	Bits	Description
		bounds check scratch space messages. Writes out of bounds are ignored. Reads out of bounds return 0. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two.
M0.2	31:0	Global Offset. Specifies the global element offset into the buffer. For the Unaligned OWord messages, this offset is in units of Bytes but must be DWord-aligned (bits 1:0 MBZ). For the other OWord messages, this offset is in units of OWords. For the DWord messages, this offset is in units of DWords. For the Byte messages, this offset is in units of Bytes. Format = U32 Range = [0,FFFFFFFCh] for Unaliged OWord messages. Range = [0,0FFFFFFFh] for other OWord messages. Range = [0,3FFFFFFFh] for DWord messages. Range = [0,FFFFFFFFh] for Byte messages.
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Write Commit Writeback Message

The writeback message is only sent on Data Port Write messages if the **Send Write Commit Message** bit in the message descriptor is set. The destination register is not modified. Write messages without the **Send Write Commit Message** bit set will not return anything to the thread (response length is 0 and destination register is null).

DWord	Bit	Description
W0.7:0		Reserved

OWord Block ReadWrite

This message takes one offset (Global Offset), and reads or writes 1, 2, 4, or 8 contiguous OWords starting at that offset.

Restrictions

Restriction
The only surface type allowed is SURFTYPE_BUFFER.
The surface format is ignored; data is returned from the constant buffer to the GRF without format conversion.
The surface cannot be tiled.
The surface base address must be OWord-aligned.
The **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model.
The **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model.

Applications:

- Constant buffer reads of a single constant or multiple contiguous constants.
- Scratch space reads/writes where the index for each pixel/vertex is the same.
- Block constant reads, scratch memory reads/writes for media.

Execution Mask. The low 8 bits of the execution mask are used to enable the 8 channels in the first and third GRF registers returned (W0, W2) for read, or the first and third write registers sent (M1, M3). The high 8 bits are used similarly for the second and fourth registers (W1, W3 or M2, M4). For reads, any mask bit set within a group of four causes the entire OWord to be read and returned to the destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

For the 1-OWord messages, only the low 8 bits of the execution mask are used. Either the low 4 bits or the high 4 bits, depending on the position of the OWord to be read or written, are used as the single group of four with behavior following that in the preceding paragraph.

The above behavior enables a SIMD16 thread to use the 8-OWord form of this message to access two channels (red and green) of a single scratch register across 16 pixels. A second message would access the other two channels (blue and alpha). The execution mask is used to ensure that data associated with inactive pixels are not overwritten.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and do not modify memory.

Message Descriptor

Bits	Description
13	Reserved: MBZ
12	Ignored
11	Ignored
10:8	**Block Size.** Specifies the number of contiguous OWords to be read or written 000: 1 OWord, read into or written from the low 128 bits of the destination register. 001: 1 OWord, read into or written from the high 128 bits of the destination register. 010: 2 OWords 011: 4 OWords 100: 8 OWords All other encodings are reserved. **Programming Note** **Context:** Message Descriptor The 6 OWord block size is valid only with **Data Port Constant Cache**.

Message Payload (Write)

For the write operation, the message payload consists of one, two, or four registers (not including the header) depending on the **Block Size** specified in the message. For the one-constant case, data is taken from either the high or low half of the payload register depending on the half selected in **Block Size**. In this case, the other half of the payload register is ignored.

The **Offset** referred to below is the **Global Offset** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bits	Description
M1.7:4	127:0	**OWord[Offset + 1].** If the block size is 1, OWord to be written from the high 128 bits of the destination, OWord[Offset] will appear in this location.
M1.3:0	127:0	OWord[Offset]
M2.7:4	127:0	OWord[Offset+3]
M2.3:0	127:0	OWord[Offset+2]
M3.7:4	127:0	OWord[Offset+5]
M3.3:0	127:0	OWord[Offset+4]
M4.7:4	127:0	OWord[Offset+7]
M4.3:0	127:0	OWord[Offset+6]

Writeback Message (Read)

For the read operation, the writeback message consists of one, two, three, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

The **Offset** referred to below is the **Global Offset** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bits	Description
W0.7:4	127:0	**OWord[Offset + 1].** If the block size is 1, OWord to be loaded into the high 128 bits of the destination, OWord[Offset] will appear in this location.
W0.3:0	127:0	OWord[Offset]
W1.7:4	127:0	OWord[Offset+3]
W1.3:0	127:0	OWord[Offset+2]
W2.7:4	127:0	OWord[Offset+5]
W2.3:0	127:0	OWord[Offset+4]
W3.7:4	127:0	OWord[Offset+7]
W3.3:0	127:0	OWord[Offset+6]

Unaligned OWord Block Read

This message takes one DWord aligned offset (**Global Offset**), and reads 1, 2, 4, or 8 contiguous OWords starting at that offset. This message is identical to the OWord Block Read message except for the offset alignment. For read/write cache, only the read path supports this unaligned OWord Block access.

Restrictions

Restriction
The only surface type allowed is SURFTYPE_BUFFER.
The surface format is ignored; data is returned from the constant buffer to the GRF without format conversion.
The surface cannot be tiled.
The surface base address must be OWord-aligned.
The **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model.
The **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model.

Applications: Reads with an offset that is not aligned with data size, such as row store usage in media.

Execution Mask. The execution mask is ignored by this message.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0.

Message Descriptor

Bits	Description
13	Ignored
12:11	Ignored
10:8	**Block Size.** Specifies the number of contiguous OWords to be read. 000: 1 OWord, read into the low 128 bits of the destination register. 001: 1 OWord, read into the high 128 bits of the destination register. 010: 2 OWords. 011: 4 OWords. 100: 8 OWords. All other encodings are reserved.

Writeback Message (Read)

For the read operation, the writeback message consists of one, two, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

The **Global Offset** is in units of **Bytes**, aligned to **DWord** (two LSBs set to zero). The **OWordX** array in units of OWord starts at Global Offset.

DWord	Bit	Description
W0.7:4	127:0	**OWord1 = *(&OWord0 + 1).** If the block size is 1 OWord to be loaded into the high 128 bits of the destination, OWord0 will appear in this location
W0.3:0	127:0	OWord0 = Buffer[Global Offset]
W1.7:4	127:0	OWord3 = *(&OWord2 + 1)
W1.3:0	127:0	OWord2 = *(&OWord1 + 1)
W2.7:4	127:0	OWord5= *(&OWord4 + 1)
W2.3:0	127:0	OWord4 = *(&OWord3 + 1)
W3.7:4	127:0	OWord7 = *(&OWord6 + 1)
W3.3:0	127:0	OWord6 = *(&OWord5 + 1)

OWord Dual Block ReadWrite

This message takes two offsets, and reads or writes 1 or 4 contiguous OWords starting at each offset. The Global Offset is added to each of the specific offsets.

The message header is no longer required for the *OWord Dual Block Read/Write* messages if sent to the data cache data port. If header is not sent, the **Global Offset** field is assumed to be zero. The header is required, however, if the binding table index is 255 (stateless model), as the **Immediate Buffer Base Address** field is required.

Programming Restriction: Writes to overlapping addresses have undefined write ordering.

Restrictions

Restriction
The only surface type allowed is SURFTYPE_BUFFER.
The surface format is ignored; data is returned from the constant buffer to the GRF without format conversion.
The surface cannot be tiled.
The surface base address must be OWord-aligned.
The **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model.
The **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model.

Applications:

- SIMD4x2 constant buffer reads where the indices of each vertex/pixel are different (if there are two indices and they are the same, hardware will optimize the cache accesses and do only one cache access).
- SIMD4x2 scratch space reads/writes where the indices are different.

Execution Mask. The low 8 bits of the execution mask are used to enable the 8 channels in the GRF registers returned for read, or each of the write registers sent. For reads, any mask bit asserted within a group of four will cause the entire OWord to be read and returned to the destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and do not modify memory contents.

Message Descriptor

Bits	Description
13	Reserved: MBZ
12:10	Ignored
9:8	**Block Size.** Specifies the number of OWords in each block to be read or written: 00: 1 OWord 10: 4 OWords All other encodings are reserved.

Message Payload

DWord	Bits	Description
M1.7	31:0	Ignored
M1.6	31:0	Ignored
M1.5	31:0	Ignored
M1.4	31:0	Block Offset 1. Specifies the OWord offset of OWord Block 1 into the surface. Format = U32 Range = [0,0FFFFFFFh]
M1.3	31:0	Ignored
M1.2	31:0	Ignored
M1.1	31:0	Ignored
M1.0	31:0	Block Offset 0

Additional Message Payload (Write)

For the write operation, the message payload consists of one or four registers (not including the header or the first part of the payload) depending on the **Block Size** specified in the message.

The **Offset1/0** referred to below is the **Global Offset** added to the corresponding **Block Offset 1/0** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bit	Description
M2.7:4	127:0	OWord[Offset1]
M2.3:0	127:0	OWord[Offset0]
M3.7:4	127:0	OWord[Offset1+1]
M3.3:0	127:0	OWord[Offset0+1]
M4.7:4	127:0	OWord[Offset1+2]
M4.3:0	127:0	OWord[Offset0+2]
M4.7:4	127:0	OWord[Offset1+3]
M4.3:0	127:0	OWord[Offset0+3]

Writeback Message (Read)

For the read operation, the writeback message consists of one or four registers depending on the **Block Size** specified in the message.

The **Offset1/0** referred to below is the **Global Offset** added to the corresponding **Block Offset 1/0** and is in units of OWords. The **OWord** array index is also in units of OWords.

DWord	Bits	Description
W0.7:4	127:0	OWord[Offset1]
W0.3:0	127:0	OWord[Offset0]
W1.7:4	127:0	OWord[Offset1+1]
W1.3:0	127:0	OWord[Offset0+1]
W2.7:4	127:0	OWord[Offset1+2]
W2.3:0	127:0	OWord[Offset0+2]
W3.7:4	127:0	OWord[Offset1+3]
W3.3:0	127:0	OWord[Offset0+3]

Media Block ReadWrite

The read form of this message enables a rectangular block of data samples to be read from the source surface and written into the GRF. The write form enables data from the GRF to be written to a rectangular block.

Restrictions

Restriction
The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Because of this, the stateless surface model is not supported with this message.
Accesses are allowed to SURFTYPE_NULL, reads return 0 and writes are ignored.
The surface format is used to determine the pixel structure for boundary clamp; the raw data from the surface is returned to the thread without any format conversion nor filtering operation.
The target cache cannot be the data cache.
The surface base address must be 32-byte aligned.
When a surface is XMajor tiled, (**tilewalk** field in the surface state is set to TILEWALK_XMAJOR), a memory area mapped through the Render Cache cannot be read and/or written in mixed frame and field modes. For example, if a memory location is first written with a zero Vertical Line Stride (frame mode), and later on (without render cache flush) read back using Vertical Line Stride of one (field mode), the read data stored in the GRF are uncertain.
The block width and offset should be aligned to the size of pixels stored in the surface. For a surface with 8bpp pixels for example, the block width and offset can be byte-aligned. For a surface with 16bpp pixels, it is word-aligned.
For YUV422 formats, the block width and offset must be pixel pair aligned (i.e. DWord-aligned).
The write form of this message has the additional restriction that both **X Offset** and **Block Width** must be

Restriction
DWord-aligned.
Pitch must be a multiple of 64 bytes when the surface is linear.
Media block writes to linear or tileX surfaces must have a height of 16 or less.

Applications: Block reads/writes for media.

Execution Mask. The execution mask on the *send* instruction for this type of message is ignored. The data that is read or written is determined completely by the block parameters.

Out-of-Bounds Accesses. Reads outside of the surface results in the address being clamped to the nearest edge of the surface and the pixel in the position being returned. Writes outside of the surface are dropped and will not modify memory contents.

Determining the boundary pixel value depends on the surface format. Surface format definitions can be found in the Surface Formats Section of the Sampling Engine Chapter.

For a surface with 8bpp pixels, the boundary byte is replicated. For example, for a boundary DWord B0B1B2B3, to replicate the left boundary byte pixel, the out of bound DWords have the format B0B0B0B0, and the format for the right boundary is B3B3B3B3.

This rule applies to all surface formats with BPE of 8. As the data port does not perform format conversion, the most likely used surface formats are R8_UINT and R8_SINT.

For any other surfaces with 16bpp pixels, boundary pixel replication is on words. For example, for a boundary dword B0B1B2B3, to replicate the left boundary word pixel, the out of bound DWords have the format B0B1B0B1, and the format for the right boundary is B2B3B2B3.

This rule applies to all surface formats with BPE of 16. As the data port does not perform format conversion, only the formats with integer data types may be useful in practice.

For a boundary DWord Y0U0Y1V0, to replicate the left boundary, we get Y0U0**Y0**V0, and to replicate the right boundary, we get **Y1**U0Y1V0.

For a boundary DWord U0Y0V0Y1, to replicate the left boundary, we get U0Y0V0**Y0**, and to replicate the right boundary, we get U0**Y1**V0Y1.

For a surface with 32bpp pixels, the boundary DWord pixel is replicated.

This rule applies to all surface formats with BPE of 32. As the data port does not perform format conversion, some of the formats may not be useful in practice.

Hardware behavior for any other surface types is undefined.

NV12 surface state: The width of the surface should be always multiples of 4 pixels. For 16bpp input message (422 8-bit) the width will always need to be in multiples of 8 bytes and for 32bpp input message (422 16-bit or 444 8-bit) the width should be in multiples of 16 bytes. Height should be in multiples of 2 pixels high. (Presently the MFX restriction is that width should be in multiples of 2 pixels.)

The y-offset of the media block write from the EU should always be even.

The x-offset of the media block write from the EU should be in multiples of 4 pixels.

The media block DWord write can have only the following combinations (when NV12 output format is used):

- 8 pixels wide for 422 8-bit mode
- 4 pixels wide for 422 8-bit mode
- 4 pixels wide for 422 16-bit
- 4 pixels wide for 444 8-bit
- 444 16-bit input format cannot be supported when the output format is NV12 (SW should not use this combination).
- It has to be in multiples of 2 pixels high for all above modes.

If 444-format is used then we use only the pixel_0 UV values of the 2x2 pixel and the rest are dropped and in case of 422-format the top UV values are used and the bottom UV valuesare dropped if the output format is NV12 format.

Message Descriptor

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>13</td><td>Reserved: MBZ</td></tr>
<tr><td>12</td><td>Reserved: MBZ</td></tr>
<tr><td>11</td><td>Reserved: MBZ</td></tr>
<tr><td>10</td><td>Vertical Line Stride Override

Specifies whether the Vertical Line Stride and Vertical Line Stride Offset fields in the surface state should be replaced by bits 9 and 8 below.

If this field is 1, Height in the surface state (see SURFACE_STATE section of Sampling Engine chapter) is modified according the following rules:
<table>
<tr><th>Vertical Line Stride (in surface state)</th><th>Override Vertical Line Stride</th><th>Derived 1-based Surface Height (As a function of the 0-based Height in Surface State)</th></tr>
<tr><td>0</td><td>0</td><td>Height + 1
(Normal)</td></tr>
<tr><td>0</td><td>1</td><td>(Height +1) / 2
Restriction: (Height + 1) must be an even number.</td></tr>
<tr><td>1</td><td>0</td><td>(Height + 1) * 2</td></tr>
<tr><td>1</td><td>1</td><td>Height + 1
(Normal)</td></tr>
</table>
</td></tr>
</table>

Bit	Description
	For example, for a 720x480 standard resolution video buffer, if Vertical Line Stride in surface state is 0, i.e. a frame, Height (of the frame) should be 479. When accessing the bottom field of this frame video buffer, if both Override Vertical Line Stride and Override Vertical Line Stride Offset are set to 1, then the derived surface height (of the field) is 240 ((Height + 1) / 2). In contrast, if Vertical Line Stride in surface state is 1 and Vertical Line Stride Offset in surface state is 0, the surface state represents the top field of the video buffer. In this case, Height (of the top field) should be programmed as 239. Accessing the bottom video field uses the same surface height of 240. Accessing the video frame (with Override Vertical Line Stride and Override Vertical Line Stride Offset of 0) results in a derived surface height of 480 ((Height + 1) * 2). 0: Use parameters in the surface state and ignore bits 9:8. 1: Use bits 9:8 to provide the Vertical Line Stride and Vertical Line Stride Offset.
9	**Override Vertical Line Stride** Specifies number of lines (0 or 1) to skip between logically adjacent lines – provides support of interleaved (field) surfaces as textures. Format = U1 in lines to skip between logically adjacent lines.
8	**Override Vertical Line Stride Offset** Specifies the offset of the initial line from the beginning of the buffer. Ignored when **Override VerticalLine Stride** is 0. Format = U1 in lines of initial offset (when Vertical Line Stride == 1).

Message Header

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td rowspan="2">M0.5</td><td>31:8</td><td>Ignored</td></tr>
<tr><td>7:0</td><td>FFTID. This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.</td></tr>
<tr><td>M0.4</td><td>31:0</td><td>Ignored (reserved for hardware delivery of binding table pointer)</td></tr>
<tr><td>M0.3</td><td>31:0</td><td>Pixel Mask. One bit per pixel (each pixel being a DWord) indicating which pixels are to be written. This field is ignored by the read message, all pixels are always returned. This field is used only if the Message Mode field is set to PIXEL_MASK. The bits in this mask correspond to the pixels (DWords) as follows:
<table><tr><td>0</td><td>1</td><td>4</td><td>5</td><td>16</td><td>17</td><td>20</td><td>21</td></tr><tr><td>2</td><td>3</td><td>6</td><td>7</td><td>18</td><td>19</td><td>22</td><td>23</td></tr><tr><td>8</td><td>9</td><td>12</td><td>13</td><td>24</td><td>25</td><td>28</td><td>29</td></tr><tr><td>10</td><td>11</td><td>14</td><td>15</td><td>26</td><td>27</td><td>30</td><td>31</td></tr></table>Byte Mask One bit per byte indicating which bytes are to be written. This field is ignored by the read message; all bytes are always returned. This field is used only if the Message Mode is BYTE_MASK. The Byte Mask is a horizontal mask applied to each row of output.
<table><tr><td>0</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td><td>7</td></tr><tr><td>8</td><td>9</td><td>10</td><td>11</td><td>12</td><td>13</td><td>14</td><td>15</td></tr><tr><td>16</td><td>17</td><td>18</td><td>19</td><td>20</td><td>21</td><td>22</td><td>23</td></tr><tr><td>24</td><td>25</td><td>26</td><td>27</td><td>28</td><td>29</td><td>30</td><td>31</td></tr></table></td></tr>
<tr><td rowspan="3">M0.2</td><td>31:30</td><td>Message Mode. This field selects the mode of this message as follows:
00: NORMAL. The Block Height and Block Width fields are set in M0.2. The Pixel Mask is not explicitly set but behaves as if it is set to all ones.
01: PIXEL_MASK: The Pixel Mask field is set in M0.3. The Block Height and Block Width are not explicitly set but behave as if they are set to 4 rows and 32 bytes, respectively.
10: BYTE_MASK: The Block Height and Block Width fields are set in M0.2. The Byte Mask in M0.3 is used to qualify which bytes are written.
11: Reserved.
For the Sampler Cache Data Port, this field is also ignored, behaving as if always set to NORMAL.</td></tr>
<tr><td>29</td><td>Ignored</td></tr>
<tr><td>28:24</td><td>Sub-Register Offset. This field provides the sub-register offset in unit of byte of a media block read message. This field is ignored (reserved) for a media block write message.
Programming Notes:
Sub-Register Offset must be aligned to BasePitch (therefore will be a multiple of DWords as</td></tr>
</table>

DWord	Bits	Description
		well). When **Register Pitch Control** = 0, **Sub-Register Offset** must align to **BasePitch*Block Height** and the output fits in a single GRF register. In general (and specifically when **Sub-Register Offset** is greater than 0), when the resulting data cross a GRF register boundary, the data must be placed symmetrically between GRF registers. **Sub-Register Offset** and **Register Pitch Control** allow software to assemble multiple media block reads directly into a shared GRF register set. For example, if both are set to zero, the read data are written to GRF registers, aligning to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the **Block Width**. If **Register Pitch Control** is non-zero, multiple media block read messages sharing the same **Register Pitch Control** but with different **Sub-Register Offset** can fill in the same set of GRF registers with media block data line interleaved. This field must be zero for Render Cache Data Port. Format = U5 Range = [0, 28] (Only a multiple of **BasePitch**, including 0, is valid.)
	21:16	**Block Height.** Height in rows of block being accessed. **Programming Note:** The Block Height is restricted to the following maximum values depending on the Block Width: **Block Width (bytes)** / **Maximum Block Height (rows)** 1-4 / 64 5-8 / 32 9-16 / 16 17-32 / 8 33-64 / 4 **Programming Note:** Block width > 32 bytes is allowed only for media block write messages and only for linear and Tile X surfaces. Format = U6 Range = [0,63] representing 1 to 64 rows
	15:10	Ignored
	9:8	**Register Pitch Control.** This field controls the register pitch for a media block read message. **Register Pitch Control** is only allowed to be non-zero if **Block Width** is a multiple of DWords. The effective register pitch must be less than or equal to 32 bytes (to fit in a single GRF register). Defining **BasePitch** as the next power-of-2 that is greater than or equal to the **Block Width, Register Pitch Control** set the register pitch in terms of **BasePitch** as the following.

DWord	Bits	Description
		Range = [0,3] representing 1 to 4 **BasePitch** **Programming Notes:** This field must be zero for Render Cache Data Port. This field is reserved (must be zero)) for a media block write message. **Programming Restriction:** This field must be zero.
	7:6	Ignored
	5:0	**Block Width.** Width in bytes of the block being accessed. **Programming Note:** Must be DWord-aligned for the write form of the message. Format = U6 Range = [0,63] representing 1 to 64 bytes
M0.1	31:0	**Y offset.** The Y offset of the upper left corner of the block into the surface. Format = S31 **Programming Note:** If **Message Mode** is set to PIXEL_MASK, this field must be a multiple of 4.
M0.0	31:0	**X offset.** The X offset of the upper left corner of the block into the surface. Must be DWord-aligned (Bits 1:0 MBZ) for the write form of the message. The **X offset** field defines the offset in the input message block. This may differ from the offset in the surface if Color Processing is enabled due to format conversion. Format = S31 **Programming Note:** If **Message Mode** is set to PIXEL_MASK, this field must be a multiple of 32.

The legal combinations of block width, pitch control, sub-register offset, and block height are given below:

Block Height for given block width, pitch control, subreg offsets									
		sub-register offsets							
block width	pitch control	0	1	2	3	4	5	6	7
1-4	00	1-64	1	1	1	1	1	1	1
	01	1-64	1-64	illegal	illegal	1-2	1-2	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-64	1-64	1-64	1-64	illegal	illegal	illegal	illegal
5-8	00	1-32	illegal	1	illegal	1	illegal	1	illegal
	01	1-32	illegal	1-32	illegal	illegal	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-32	illegal	1-32	illegal	1-32	illegal	1-32	illegal
9-16	00	1-16	illegal	illegal	illegal	1	illegal	illegal	illegal
	01	1-16	illegal	illegal	illegal	1-16	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-16	illegal	illegal	illegal	1-16	illegal	illegal	illegal
17-32	00	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	01	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	10	illegal	illegal	illegal	illegal	illegal	illegal	illegal	illegal
	11	1-8	illegal	illegal	illegal	illegal	illegal	illegal	illegal

Message Payload (Write)

DWord	Bit	Description
M1:n		**Write Data.** The format of the write data depends on the **Block Height** and **Block Width**. The data is aligned to the least significant bits of the first register, and the register pitch is equal to the next power-of-2 that is greater than or equal to the **Block Width**.

If **Color Processing Enable** is enabled, the write data is divided into pixels according to the **Message Format** field. The fields within each pixel are defined below. For the 4:2:2 modes, each pixel position includes channels for two pixels.

Message Format	31:24	23:16	15:8	7:0
YUV 4:2:2, 8 bits per channel	Cr (V)	right pixel lum (Y1)	Cb (U)	left pixel lum (Y0)
YUV 4:4:4, 8 bits per channel	alpha (A)	luminance (Y)	Cb (U)	Cr (V)
	63:48	47:32	31:16	15:0
YUV 4:2:2, 16 bits per channel	Cr (V)	right pixel lum (Y1)	Cb (U)	left pixel lum (Y0)
YUV 4:4:4, 16 bits per channel	alpha (A)	Cr (V)	luminance (Y)	Cb (U)

Writeback Message (Read)

DWord	Bits	Description
W0:n	31:0	**Read Data.** The format of the read data depends on the **Block Height**, **Block Width**, **Register Pitch Control**, and **Sub-Register Offset**. The data is aligned to the **Sub-Register Offset** of the first register, and the register pitch is set to one or more **BasePatch**.

DWord Scattered ReadWrite

This message takes a set of offsets, and reads or writes 8 or 16 scattered DWords starting at each offset. The Global Offset is added to each of the specific offsets.

Restrictions

Restriction
The only surface type allowed is SURFTYPE_BUFFER.
The surface format is ignored; data is returned from the constant buffer to the GRF without format conversion.
The surface cannot be tiled.
The surface base address must be DWord-aligned.
Writes to overlapping addresses have undefined write ordering.
For read messages with X/Y offsets that are outside the bounds of the surface, the address is clamped to the nearest edge of the surface. For write messages with X/Y offsets that are outside the bounds of the surface, the behavior is undefined.
The **Render Cache Read Write Mode** field in SURFACE_STATE must be set to read/write mode when using this message with the render cache in the surface state model.
The **Stateless Render Cache Read-Write Mode** field in the SVG_WORK_CTL register must be set to read/write mode when using this message with the render cache in the stateless model.
Hardware does check for and optimize for cases where offsets are equal or contiguous, however for optimal performance in some of these cases a different message may provide higher performance.
The message header is no longer required for the *OWord DWord Scattered Read/Write* messages if sent to the data cache data port. If header is not sent, the **Global Offset** field is assumed to be zero. The header is required, however, if the binding table index is 255 (stateless model), as the **Immediate Buffer Base Address** field is required.

Applications:

- SIMD8/16 constant buffer reads where the indices of each pixel are different (read one channel per message)
- SIMD8/16 scratch space reads/writes where the indices are different (read/write one channel per message)
- General purpose DWord scatter/gathering, used by media

Execution Mask. Depending on the block size, either the low 8 bits or all 16 bits of the execution mask are used to determine which DWords are read into the destination GRF register (for read), or which DWords are written to the surface (for write).

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

Message Descriptor

Bits	Description
13	Reserved: MBZ
12	Ignored
11:10	Ignored
9:8	**Block Size.** Specifies the number of DWords read or written: 10: 8 DWords 11: 16 DWords All other encodings are reserved.

Message Payload

DWord	Bits	Description
M1.7	31:0	Offset 7. Specifies the DWord offset of DWord 7 into the surface. Format = U32 Range = [0,3FFFFFFFh]
M1.6	31:0	Offset 6
M1.5	31:0	Offset 5
M1.4	31:0	Offset 4
M1.3	31:0	Offset 3
M1.2	31:0	Offset 2
M1.1	31:0	Offset 1
M1.0	31:0	Offset 0
M2.7	31:0	**Offset 15.** This message register is included only if the block size is 16 DWords.
M2.6	31:0	Offset 14
M2.5	31:0	Offset 13
M2.4	31:0	Offset 12
M2.3	31:0	Offset 11
M2.2	31:0	Offset 10
M2.1	31:0	Offset 9
M2.0	31:0	Offset 8

Additional Message Payload (Write)

For the write operation, either one or two additional registers (depending on the block size) of payload contain the data to be written.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M3.7	31:0	DWord[Offset7]
M3.6	31:0	DWord[Offset6]
M3.5	31:0	DWord[Offset5]
M3.4	31:0	DWord[Offset4]
M3.3	31:0	DWord[Offset3]
M3.2	31:0	DWord[Offset2]
M3.1	31:0	DWord[Offset1]
M3.0	31:0	DWord[Offset0]
M4.7	31:0	**DWord[Offset15].** This message register is included only if the block size is 16 DWords
M4.6	31:0	DWord[Offset14]
M4.5	31:0	DWord[Offset13]
M4.4	31:0	DWord[Offset12]
M4.3	31:0	DWord[Offset11]
M4.2	31:0	DWord[Offset10]
M4.1	31:0	DWord[Offset9]
M4.0	31:0	DWord[Offset8]

Writeback Message (Read)

For the read operation, the writeback message consists of either one or two registers depending on the block size.

The **DWord** array index is also in units of DWords.

DWord	Bits	Description
W0.7	31:0	DWord[Offset7]
W0.6	31:0	DWord[Offset6]
W0.5	31:0	DWord[Offset5]
W0.4	31:0	DWord[Offset4]
W0.3	31:0	DWord[Offset3]
W0.2	31:0	DWord[Offset2]
W0.1	31:0	DWord[Offset1]
W0.0	31:0	DWord[Offset0]

DWord	Bits	Description
W1.7	31:0	**DWord[Offset15].** This writeback message register is included only if the block size is 16 DWords.
W1.6	31:0	DWord[Offset14]
W1.5	31:0	DWord[Offset13]
W1.4	31:0	DWord[Offset12]
W1.3	31:0	DWord[Offset11]
W1.2	31:0	DWord[Offset10]
W1.1	31:0	DWord[Offset9]
W1.0	31:0	DWord[Offset8]

Message Descriptor

Bit	Description
12	**Two-Source Message**. When this bit is set, there are two data-phases for two sources. Two-source message is used only for opcode "0111" and for all other opcodes this bit must be 0. When this bit is 0, M3 is not sent to the data-port.
11:8	Atomic Operation Code: (Please refer to the table below) Unsupported opcodes: 1101, 1110, 1111

Opcode	Operation	Return Value
0000	ADD: new = old + src0	Old value
0001	SUB: new = old – src0	Old value
0010	INC : new = old+1	Old value
0011	DEC: new = old-1	Old value
0100	MIN: new = min(old, src0)	Old value
0101	MAX: new = max(old, src0)	Old value
0110	XCHG: new = src0	Old value
0111	CMPXCHG : new = (old==src1) ? src0 : old	Old value
1000	AND: new = old & src0	Old value
1001	OR: new = old | src0	Old value
1010	XOR: new = old ^ src0	Old value
1011	MIN_SINT: new = min(old, src0)	Old value(signed)
1100	MAX_SINT: new = max(old, src0)	Old value(signed)
1101-1111		Old value

Message Payload

DWord	Bit	Description
M1.7	31:0	Offset 7. Specifies the DWord offset of DWord 7 into the surface. Format = U32 Range = [0,3FFFFFFFh]
M1.6	31:0	Offset 6
M1.5	31:0	Offset 5
M1.4	31:0	Offset 4
M1.3	31:0	Offset 3
M1.2	31:0	Offset 2
M1.1	31:0	Offset 1
M1.0	31:0	Offset 0

Source Payload

Either one or two additional registers (depending on **Two-Source Message**) of source payload contain the data to be used as source.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M2.7	31:0	DWord[Offset7] Src0
M2.6	31:0	DWord[Offset6] Src0
M2.5	31:0	DWord[Offset5] Src0
M2.4	31:0	DWord[Offset4] Src0
M2.3	31:0	DWord[Offset3] Src0
M2.2	31:0	DWord[Offset2] Src0
M2.1	31:0	DWord[Offset1] Src0
M2.0	31:0	DWord[Offset0] Src0
M3.7	31:0	DWord[Offset7] Src1
M3.6	31:0	DWord[Offset6] Src1
M3.5	31:0	DWord[Offset5] Src1
M3.4	31:0	DWord[Offset4] Src1
M3.3	31:0	DWord[Offset3] Src1
M3.2	31:0	DWord[Offset2] Src1
M3.1	31:0	DWord[Offset1] Src1
M3.0	31:0	DWord[Offset0] Src1

Writeback Message

For the read operation, the writeback message consists of either one or two registers depending on the block size.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
W0.7	31:0	DWord[Offset7]
W0.6	31:0	DWord[Offset6]
W0.5	31:0	DWord[Offset5]
W0.4	31:0	DWord[Offset4]
W0.3	31:0	DWord[Offset3]
W0.2	31:0	DWord[Offset2]
W0.1	31:0	DWord[Offset1]

DWord	Bit	Description
W0.0	31:0	DWord[Offset0]

Byte Scattered ReadWrite

These messages are supported on CHV, BSW only.

These messages take a set of offsets, and read or write 8 or 16 scattered and possibly misaligned bytes, words, or DWords starting at each offset. The **Global Offset** from the message header is added to each of the specific offsets.

Restrictions

Restriction
The only surface type allowed is SURFTYPE_BUFFER.
The surface format is ignored; data is returned from the buffer to the GRF without format conversion.
The surface cannot be tiled.
The surface base address must be DWord-aligned.
Writes to overlapping addresses have undefined write ordering.
The surface is treated as a 1-dimensional surface. The element size (pitch) times the number of elements is used to determine the size of the buffer for out-of-bounds checking if using the surface state model. Out of bounds checking is done at DWord granularity; if any part of the DWord is out-of-bounds then the whole DWord is considered out-of-bounds.
The stateless model is supported. Bounds checking for a stateless message is 4GB overflow and < General State upper bound.

Applications: Byte aligned buffer accesses in GPGPU programs.

Execution Mask. Depending on the block size, either the low 8 bits or all 16 bits of the execution mask are used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write).

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

Message Descriptor

Bit	Description
13:12	Ignored
11:10	**Data Size.** Specifies the data size for each slot. 0: 1 byte 1: 2 bytes 2: 4 bytes 3: Reserved
9	Ignored
8	**SIMD Mode.** Specifies the SIMD mode of the message (number of slots processed). 0: SIMD8 1: SIMD16

Message Payload

DWord	Bits	Description
M1.7	31:0	Offset 7. Specifies the byte offset of DWord 7 into the surface. Format = U32 Range = [0,FFFFFFFFh]
M1.6	31:0	Offset 6
M1.5	31:0	Offset 5
M1.4	31:0	Offset 4
M1.3	31:0	Offset 3
M1.2	31:0	Offset 2
M1.1	31:0	Offset 1
M1.0	31:0	Offset 0
M2.7	31:0	**Offset 15.** This message register is included only if the **SIMD Mode** is SIMD16.
M2.6	31:0	Offset 14
M2.5	31:0	Offset 13
M2.4	31:0	Offset 12
M2.3	31:0	Offset 11
M2.2	31:0	Offset 10
M2.1	31:0	Offset 9
M2.0	31:0	Offset 8

Additional Message Payload (Write)

For the write operation, either one or two additional registers (depending on the block size) of payload contain the data to be written.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of bytes. The length of **Data** written depends on the **Data Size** and is right-justified within the 32-bit field. The upper bits are ignored for 1 byte and 2 byte **Data Size**.

DWord	Bit	Description
M3.7	31:0	Data[Offset7]
M3.6	31:0	Data[Offset6]
M3.5	31:0	Data[Offset5]
M3.4	31:0	Data[Offset4]
M3.3	31:0	Data[Offset3]
M3.2	31:0	Data[Offset2]

DWord	Bit	Description
M3.1	31:0	Data[Offset1]
M3.0	31:0	Data[Offset0]
M4.7	31:0	**Data[Offset15].** This message register is included only if the **SIMD Mode** is SIMD16.
M4.6	31:0	Data[Offset14]
M4.5	31:0	Data[Offset13]
M4.4	31:0	Data[Offset12]
M4.3	31:0	Data[Offset11]
M4.2	31:0	Data[Offset10]
M4.1	31:0	Data[Offset9]
M4.0	31:0	Data[Offset8]

Writeback Message (Read)

For the read operation, the writeback message consists of either one or two registers depending on the block size.

The **Offsetn** referred to below is the **Global Offset** added to the corresponding **Offset n** and is in units of bytes. The length of **Data** written depends on the **Data Size** and is right-justified within the 32-bit field and only the requeted bytes are written to the GRF.

DWord	Bit	Description
W0.7	31:0	Data[Offset7]
W0.6	31:0	Data[Offset6]
W0.5	31:0	Data[Offset5]
W0.4	31:0	Data[Offset4]
W0.3	31:0	Data[Offset3]
W0.2	31:0	Data[Offset2]
W0.1	31:0	Data[Offset1]
W0.0	31:0	Data[Offset0]
W1.7	31:0	**Data[Offset15].** This message register is included only if the **SIMD Mode** is SIMD16.
W1.6	31:0	Data[Offset14]
W1.5	31:0	Data[Offset13]
W1.4	31:0	Data[Offset12]
W1.3	31:0	Data[Offset11]
W1.2	31:0	Data[Offset10]
W1.1	31:0	Data[Offset9]
W1.0	31:0	Data[Offset8]

TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation

Six data port messages (Typed Surface Read, Typed Surface Write, Typed Atomic Operation, Untyped Surface Read, Untyped Surface Write, and Untyped Atomic Operation) allow direct read/write accesses to surfaces. These messages support three major categories of surfaces:

Typed surfaces. These surfaces are of type SURFTYPE_1D, 2D, 3D, or BUFFER and have a supported surface format other than RAW.

A typed surface uses U, V, R, and LOD address parameters (the number of parameters used depends on the surface type), and performs conversion of type to/from the selected surface format as follows:

- Surface formats with UINT require the message data in U32 format.
- Surface formats with SINT require the message data in S32 format.
- All other surface formats require the message data in FLOAT32 format.

The untyped surface categories, both of which use the RAW surface format, perform no type conversion. A raw buffer uses just the U address parameter, which specifies the byte offset into the surface, which must be a multiple of 4. A structured buffer uses the U address parameter as an array index and the V address parameter as a byte offset into the array element (which also must be a multiple of 4).

For both raw and structured buffers, up to 4 DWords are accessed beginning at the byte address determined. These 4 dwords correspond to the red, green, blue, and alpha channels in that order with red mapping to the lowest order DWord. The atomic operation messages only access the first DWord (corresponding to the red channel for typed messages).

The atomic operation messages cause atomic read-modify-write operations on the “destination” location addressed. In the table below, the new value of the destination (new_dst) is computed as indicated based on the old value of the destination (old_dst) and up to two sources included in the message (src0 and src1). Optionally, a value can be returned by the message (ret).

The atomic operations guarantee that the read and the write are performed atomically, meaning that no read or write to the same memory location from this thread or any other thread can occur between the read and the write.

The following atomic operations are available, along with the specific operation performed for each and the return value:

Atomic Operation	new_dst	ret
AOP_AND	old_dst & src0	old_dst
AOP_OR	old_dst | src0	old_dst
AOP_XOR	old_dst ^ src0	old_dst
AOP_MOV	src0	old_dst
AOP_INC	old_dst + 1	old_dst
AOP_DEC	old_dst – 1	old_dst
AOP_ADD	old_dst + src0	old_dst

Atomic Operation	new_dst	ret
AOP_SUB	old_dst – src0	old_dst
AOP_REVSUB	src0 – old_dst	old_dst
AOP_IMAX	imax(old_dst, src0)	old_dst
AOP_IMIN	imin(old_dst, src0)	old_dst
AOP_UMAX	umax(old_dst, src0)	old_dst
AOP_UMIN	umin(old_dst, src0)	old_dst
AOP_CMPWR	(src0 == old_dst) ? src1 : old_dst	old_dst
AOP_PREDEC	old_dst – 1	new_dst
AOP_CMPWR8B	(src08B == old_dst8B) ? src18B : old_dst8B	old_dst8B

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
src08B is 8 bytes, src18B is 8 bytes, and old_dst8B is 8 bytes in length.	

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
AOP_CMPWR8B is not supported for SLM.	

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
AOP_CMPWR8B addresses must be QWord-aligned.	

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
imax/imin assume operands are signed integers, umax/umin assume operands are unsigned integers. All other operations treat all values as 32-bit unsigned integers. Add and subtract operations wrap without any special indication.	

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
For untyped messages, the **Tile Mode** must be LINEAR. For untyped messages, the **Surface Format** must be RAW and the **Surface Type** must be SURFTYPE_BUFFER or SURFTYPE_STRBUF. For typed messages, the **Surface Type** must be SURFTYPE_1D, 2D, 3D, or BUFFER.	

Surface Format for Typed Surface Reads

Surface Format Name
R16G16B16A16_UINT
R8G8B8A8_UINT
R16G16_UINT
R32_SINT
R32_UINT
R32_FLOAT
R8G8_UINT
R16_UINT
R8_UINT

Surface Format for Typed Surface Writes

Surface Format Name
R32G32B32A32_FLOAT
R32G32B32A32_SINT
R32G32B32A32_UINT
R16G16B16A16_UNORM
R16G16B16A16_SNORM
R16G16B16A16_SINT
R16G16B16A16_UINT
R16G16B16A16_FLOAT
R32G32_FLOAT
R32G32_SINT
R32G32_UINT
B8G8R8A8_UNORM
R10G10B10A2_UNORM
R10G10B10A2_UINT
R8G8B8A8_UNORM
R8G8B8A8_SNORM
R8G8B8A8_SINT
R8G8B8A8_UINT
R16G16_UNORM
R16G16_SNORM
R16G16_SINT
R16G16_UINT
R16G16_FLOAT

Surface Format Name
B10G10R10A2_UNORM
R11G11B10_FLOAT
R32_SINT
R32_UINT
R32_FLOAT
B5G6R5_UNORM
B5G5R5A1_UNORM
B4G4R4A4_UNORM
R8G8_UNORM
R8G8_SNORM
R8G8_SINT
R8G8_UINT
R16_UNORM
R16_SNORM
R16_SINT
R16_UINT
R16_FLOAT
B5G5R5X1_UNORM
R8_UNORM
R8_SNORM
R8_SINT
R8_UINT
A8_UNORM

General Restrictions

For typed surface writes where the Surface Format has components that are not byte-aligned, each shader channel select in the surface state must be set to a unique surface channel (SCS_RED, SCS_GREEN, SCS_BLUE, SCS_ALPHA) and the value of (SCS_ZERO, SCS_ONE) cannot be selected. Also all channels must be enabled for writing.

The **Surface Format** for typed atomic operations must be R32_UINT or R32_SINT.

For atomic operations, each shader channel select in the surface state must be set to the same surface channel (R = SCS_RED, G = SCS_GREEN, B = SCS_BLUE, A = SCS_ALPHA).

For untyped messages accessing SURFTYPE_STRBUF, the V address (byte offset) must be DWord-aligned (low 2 bits must be zero).

For untyped messages accessing SURFTYPE_BUFFER, the U address (byte offset) must be DWord-aligned (low 2 bits must be zero).

Typed messages only support SIMD8.

Project-Specific Restrictions

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
The stateless model support is limited to untyped messages. Furthermore, they are treated as SURFTYPE_BUFFER and **Surface Format** of RAW. The bounds checking for the stateless message is 4GB overflow and < General State upper bound.	

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
Tile W surfaces must be of format R8_UINT and only support SIMD8. Furthermore, only the RED channel can be enabled.	

Execution Mask

SIMD16: The 16 bits of the execution mask are ANDed with the 16 bits of the **Pixel/Sample Mask** from the message header and the resulting mask is used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write). If the header is not present, only the execution mask is used.

SIMD8: The low 8 bits of the execution mask are ANDed with 8 bits of the **Pixel/Sample Mask** from the message header. For the typed messages, the **Slot Group** in the message descriptor selects either the low or high 8 bits. For the untyped messages, the low 8 bits are always selected. The resulting mask is used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write). If the header is not present, only the low 8 bits of the execution mask are used.

SIMD4x2: Each group of 4 bits within the low 8 bits of the execution mask are ORed together to create two bits that are used to determine which slots are read into the destination GRF register.

Out–of–Bounds Accesses: Reads to areas outside of the surface return 0, except for the *Typed Surface Read* message that returns 1 in the alpha channel and 0 in the other channels. Writes to areas outside of the surface are dropped and will not modify memory contents.

Programming Note	
Context:	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation
Writes to overlapping addresses have undefined write ordering.	

SIMD Mode, Surface Category, and Message Type Combinations Supported

SIMD Mode	Surface Category	Message Type
SIMD16	Untyped	Read
SIMD16	Untyped	Write
SIMD16	Untyped	Atomic
SIMD8	Untyped	Read
SIMD8	Untyped	Write

SIMD Mode	Surface Category	Message Type
SIMD8	Untyped	Atomic
SIMD8	Typed	Read
SIMD8	Typed	Write
SIMD8	Typed	Atomic
SIMD4x2	Untyped	Read
SIMD4x2	Untyped	Write
SIMD4x2	Untyped	Atomic
SIMD4x2	Typed	Read
SIMD4x2	Typed	Write
SIMD4x2	Typed	Atomic

The following table indicates the hardware interpretation of each input parameter based on surface type. Parameters with blank entries are ignored by hardware if delivered.

Surface Type	"Surface Array" Field in SURFACE_STATE	U Address	V Address	R Address	LOD
SURFTYPE_1D	disabled	X pixel address			LOD
	enabled	X pixel address	array index		LOD
SURFTYPE_2D	disabled	X pixel address	Y pixel address		LOD
	enabled	X pixel address	Y pixel address	array index	LOD
SURFTYPE_3D	disabled	X pixel address	Y pixel address	Z pixel address	LOD
SURFTYPE_BUFFER	disabled	buffer index			
SURFTYPE_STRBUF	disabled	buffer index	byte offset		

Typed Surface ReadWrite Message Descriptor

Bit	Description
13	Slot Group This field controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present. Format = U1 0: Use low 8 slots 1: Use high 8 slots
12	Ignored
11	Alpha Channel Mask For the read message, indicates that alpha will be included in the writeback message. For the write message, indicates that alpha is included in the message payload, and that alpha will be written to the surface. 0: Alpha channel included 1: Alpha channel not included Programming Notes: At least one of the channels must be unmasked (the 4-bit channel mask cannot be 1111b).
10	Blue Channel Mask
9	Green Channel Mask
8	Red Channel Mask

Typed Surface Read/Write Message Descriptor

Bit	Description
13:12	Slot Group This field controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. This field is ignored if the header is not present. Format = U2 00: SIMD4x2 01: Use low 8 slots 10: Use high 8 slots 11: Reserved
11	Alpha Channel Mask For the read message, indicates that alpha will be included in the writeback message. For the write message, indicates that alpha is included in the message payload, and that alpha will be written to the surface. 0: Alpha channel included 1: Alpha channel not included Programming Notes: At least one of the channels must be unmasked (the 4-bit channel mask cannot be 1111b).
10	Blue Channel Mask
9	Green Channel Mask
8	Red Channel Mask

Untyped Surface ReadWrite Message Descriptor

Bits	Description
13:12	SIMD Mode Format = U2 0: SIMD4x2 (valid for reads & writes) 1: SIMD16 2: SIMD8 3: Reserved
11	Alpha Channel Mask For the read message, indicates that alpha will be included in the writeback message. For the write message, indicates that alpha is included in the message payload, and that alpha will be written to the surface. 0: Alpha channel included 1: Alpha channel not included Programming Notes: For the *Untyped Surface Write* message, each channel mask cannot be 0 unless all of the lower mask bits are also zero. This means that the only 4-bit channel mask values allowed are 0000b, 1000b, 1100b, and 1110b. Other messages allow any combination of channel masks. For the *Untyped Surface Read* message, at least one of the channels must be unmasked (the 4-bit channel mask cannot be 1111b).
10	Blue Channel Mask
9	Green Channel Mask
8	Red Channel Mask

Typed Atomic Operation Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	Slot Group This field controls which 8 bits of Pixel/Sample Mask in the message header are ANDed with the execution mask to determine which slots are accessed. Format = U1 0: Use low 8 slots 1: Use high 8 slots
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: Reserved 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: AOP_CMPWR 1111: AOP_PREDEC

Typed Atomic Operation SIMD4x2 Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	Reserved
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: reserved 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: AOP_CMPWR 1111: AOP_PREDEC

Untyped Atomic Operation Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	SIMD Mode Format = U1 0: SIMD16 1: SIMD8
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: 0000: AOP_CMPWR8B 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: AOP_CMPWR 1111: AOP_PREDEC

Untyped Atomic Operation SIMD4x2 Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	Reserved
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: AOP_CMPWR8B 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: AOP_CMPWR 1111: AOP_PREDEC

Atomic Counter Operation Message Descriptor

Bits	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	SIMD Mode Format: U1 0: Reserved 1: SIMD8 (low 8 slots)
11:8	Atomic Operation Type Specifies the atomic operation to perform: 0000: Reserved 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: Reserved 1111: AOP_PREDEC

For Append Counter Operations there is no address payload as the address is provided by the append counter field in the surface state. The write data payloads are the same as untyped atomic. The write backs are the same as untyped atomic. When accessing a surface with the Append Counter Operation, if the Append Counter enable field of the surface state is not 1, the access is treated as out of bounds, with writes ignored and reads returning 0.

Atomic Counter Operation SIMD4x2 Message Descriptor

Bits	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	Reserved
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: Reserved 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: Reserved 1111: AOP_PREDEC

For Append Counter Operations there is no address payload as the address is provided by the append counter field in the surface state. The write data payloads are the same as untyped atomic 4x2. The write backs are the same as untyped atomic 4x2.

When accessing a surface with the Append Counter Operation, if the Append Counter enable field of the surface state is not 1, the access is treated as out of bounds, with writes ignored and reads returning 0.

Message Header

The message header for the untyped messages only needs to be delivered for pixel shader threads, where the execution mask may indicate pixels/samples that are enabled only due to derivative (LOD) calculations, but the corresponding slot on the surface must not be accessed. Typed messages (which go to the render cache data port) must include the header.

DWord	Bits	Description
M0.7	31:16	Ignored
	15:0	**Pixel/Sample Mask.** This field contains the 16-bit pixel/sample mask to be used for SIMD16 and SIMD8 messages. All 16 bits are used for SIMD16 messages. For typed SIMD8 messages, **Slot Group** selects which 8 bits of this field are used. For untyped SIMD8 messages, the low 8 bits of this field are used. If the header is not delivered, this field defaults to all ones. The field is ignored for SIMD4x2 messages.
M0.6	31:0	Ignored
M0.5	31:0	**Immediate Buffer Base Address.** Specifies the surface base address for messages in which the Binding Table Index is 255 (stateless model), else this field is ignored. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Message Payload

The message payload consists of the following:

- For the read messages, only an address payload is delivered.
- For the write messages, an address payload is followed by the write data payload.
- For the atomic operation messages, an address payload is followed by the source payload.
- For SIMD16 and SIMD8 messages, the message length is used to determine how many address parameters are included in the message. The number of message registers in the write data payload is determined by the number of channel mask bits that are enabled, and the number of message registers in the source payload is determined by the atomic operation operation. Thus, one or neither of these two values (depending on the message type), plus one for the header, can be subtracted from the message length to determine the number of message registers in the address payload, from which the number of address parameters can be determined.

SIMD16 Address Payload

The payload of a SIMD16 message provides address parameters to process 16 slots. The possible address parameters are U and V (since SIMD16 is only supported with untyped messages). The number of parameters required depends on the surface type being accessed. Each parameter takes two message registers. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

Programming Note	
Context:	SIMD16 Address Payload
For untyped messages of surface type SURFTYPE_BUFFER, either U only can be sent or U and V can be sent. If V is sent, it is ignored.	

DWord	Bit	Description
M1.7	31:0	Slot 7 U Address Specifies the U Address for slot 7. Format = U32
M1.6	31:0	Slot 6 U Address
M1.5	31:0	Slot 5 U Address
M1.4	31:0	Slot 4 U Address
M1.3	31:0	Slot 3 U Address
M1.2	31:0	Slot 2 U Address
M1.1	31:0	Slot 1 U Address
M1.0	31:0	Slot 0 U Address
M2.7	31:0	Slot 15 U Address
M2.6	31:0	Slot 14 U Address
M2.5	31:0	Slot 13 U Address
M2.4	31:0	Slot 12 U Address
M2.3	31:0	Slot 11 U Address
M2.2	31:0	Slot 10 U Address
M2.1	31:0	Slot 9 U Address
M2.0	31:0	Slot 8 U Address
M3		Slots 7:0 V Address
M4		Slots 15:8 V Address

SIMD16 Source Payload (Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included. Message registers given here could be a lower number if some of the address parameters are not included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M5.7	31:0	Slot 7 Source0 Specifies Source0 for slot 7. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
M5.6	31:0	Slot 6 Source0
M5.5	31:0	Slot 5 Source0
M5.4	31:0	Slot 4 Source0
M5.3	31:0	Slot 3 Source0
M5.2	31:0	Slot 2 Source0
M5.1	31:0	Slot 1 Source0
M5.0	31:0	Slot 0 Source0
M6.7	31:0	Slot 15 Source0
M6.6	31:0	Slot 14 Source0
M6.5	31:0	Slot 13 Source0
M6.4	31:0	Slot 12 Source0
M6.3	31:0	Slot 11 Source0
M6.2	31:0	Slot 10 Source0
M6.1	31:0	Slot 9 Source0
M6.0	31:0	Slot 8 Source0
M7		Slots 7:0 Source1
M8		Slots 15:8 Source1

SIMD16 Source Payload (AOP_CMPWR8B Only)

DWord	Bit	Description
M5.7	31:0	Slot 7 Source0[31:0] Specifies Source0[31:0] for slot 7. Format = U32
M5.6	31:0	Slot 6 Source0[31:0]
M5.5	31:0	Slot 5 Source0[31:0]
M5.4	31:0	Slot 4 Source0[31:0]
M5.3	31:0	Slot 3 Source0[31:0]
M5.2	31:0	Slot 2 Source0[31:0]
M5.1	31:0	Slot 1 Source0[31:0]
M5.0	31:0	Slot 0 Source0[31:0]
M6.7	31:0	Slot 15 Source0[31:0]
M6.6	31:0	Slot 14 Source0[31:0]
M6.5	31:0	Slot 13 Source0[31:0]
M6.4	31:0	Slot 12 Source0[31:0]
M6.3	31:0	Slot 11 Source0[31:0]
M6.2	31:0	Slot 10 Source0[31:0]
M6.1	31:0	Slot 9 Source0[31:0]
M6.0	31:0	Slot 8 Source0[31:0]
M7		Slots 7:0 Source0[63:32]
M8		Slots 15:8 Source0[63:32]
M9		Slots 7:0 Source1[31:0]
M10		Slots 15:8 Source1[31:0]
M11		Slots 7:0 Source1[63:32]
M12		Slots 15:8 Source1[63:32]

SIMD16 Write Data Payload (Write Message Only)

The write data payload follows the address payload for write messages. Actual position within the message may vary if some of the parameters are not included or if some of the channel mask bits are asserted. Any parameter or write channel not included in the payload is skipped, with message phases below it being renumbered to take up the vacated space.

DWord	Bit	Description
M5.7	31:0	Slot 7 Red Specifies the value of the red channel to be written for slot 7. Format = 32 bits raw data.
M5.6	31:0	Slot 6 Red
M5.5	31:0	Slot 5 Red
M5.4	31:0	Slot 4 Red
M5.3	31:0	Slot 3 Red
M5.2	31:0	Slot 2 Red
M5.1	31:0	Slot 1 Red
M5.0	31:0	Slot 0 Red
M6.7	31:0	Slot 15 Red
M6.6	31:0	Slot 14 Red
M6.5	31:0	Slot 13 Red
M6.4	31:0	Slot 12 Red
M6.3	31:0	Slot 11 Red
M6.2	31:0	Slot 10 Red
M6.1	31:0	Slot 9 Red
M6.0	31:0	Slot 8 Red
M7		Slots 7:0 Green
M8		Slots 15:8 Green
M9		Slots 7:0 Blue
M10		Slots 15:8 Blue
M11		Slots 7:0 Alpha
M12		Slots 15:8 Alpha

SIMD8 Address Payload

The payload of a SIMD8 message provides address parameters to process 8 slots. The possible address parameters are U, V, R, and LOD. The number of parameters required depends on the surface type being accessed. Each parameter takes one message register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

Programming Note	
Context:	SIMD8 Address Payload.htm
• For untyped messages of surface type SURFTYPE_BUFFER, either U only can be sent or U and V can be sent. If V is sent it is ignored • For untyped messages of surface type SURFTYPE_STRBUF, both U and V must be sent.	

DWord	Bits	Description
M1.7	31:0	Slot 7 U Address Specifies the U Address for slot 7. Format = U32
M1.6	31:0	Slot 6 U Address
M1.5	31:0	Slot 5 U Address
M1.4	31:0	Slot 4 U Address
M1.3	31:0	Slot 3 U Address
M1.2	31:0	Slot 2 U Address
M1.1	31:0	Slot 1 U Address
M1.0	31:0	Slot 0 U Address
M2		Slots 7:0 V Address
M3		Slots 7:0 R Address Programming Notes: This register can only be delivered for the *Typed* message types.
M4		Slots 7:0 LOD Programming Notes: This register can only be delivered for the *Typed* message types.

SIMD8 Source Payload (Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included. Message registers given here could be a lower number if some of the address parameters are not included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M5.7	31:0	Slot 7 Source0 Specifies Source0 for slot 7. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
M5.6	31:0	Slot 6 Source0
M5.5	31:0	Slot 5 Source0
M5.4	31:0	Slot 4 Source0
M5.3	31:0	Slot 3 Source0
M5.2	31:0	Slot 2 Source0
M5.1	31:0	Slot 1 Source0
M5.0	31:0	Slot 0 Source0
M6		Slots 7:0 Source1

SIMD8 Write Data Payload (Write Message Only)

The write data payload follows the address payload for write messages. Actual position within the message may vary if some of the parameters are not included or if some of the channel mask bits are asserted. Any parameter or write channel not included in the payload is skipped, with message phases below it being renumbered to take up the vacated space.

DWord	Bit	Description
M5.7	31:0	Slot 7 Red Specifies the value of the red channel to be written for slot 7. For *Untyped* messages: Format = 32 bits raw data. For *Typed* messages: Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
M5.6	31:0	Slot 6 Red
M5.5	31:0	Slot 5 Red
M5.4	31:0	Slot 4 Red
M5.3	31:0	Slot 3 Red
M5.2	31:0	Slot 2 Red
M5.1	31:0	Slot 1 Red
M5.0	31:0	Slot 0 Red
M6		Slots 7:0 Green
M7		Slots 7:0 Blue
M8		Slots 7:0 Alpha

SIMD8 Write Data Payload (Tile W Write Message Only)

The write data payload follows the address payload for write messages. Actual position within the message may vary if some of the parameters are not included.

DWord	Bit	Description
M5.7	31:8	Ignored
	7:0	Slot 7 Red Specifies the value of the red channel to be written for slot 7. For *Typed* messages: Format = U8
M5.6	31:8	Ignored
	7:0	Slot 6 Red
M5.5	31:8	Ignored
	7:0	Slot 5 Red
M5.4	31:8	Ignored
	7:0	Slot 4 Red
M5.3	31:8	Ignored
	7:0	Slot 3 Red
M5.2	31:8	Ignored
	7:0	Slot 2 Red
M5.1	31:8	Ignored
	7:0	Slot 1 Red
M5.0	31:8	Ignored
	7:0	Slot 0 Red

SIMD4x2 Address Payload

The payload of a SIMD4x2 message provides address parameters to process 2 slots.

DWord	Bits	Description
M1.7	31:0	Slot 1 LOD **Programming Note:** This register can only be delivered for the *Typed* message types.
M1.6	31:0	Slot 1 R Address **Programming Note:** This register can only be delivered for the *Typed* message types.
M1.5	31:0	Slot 1 V Address Format = U32
M1.4	31:0	Slot 1 U Address Format = U32
M1.3	31:0	Slot 0 LOD
M1.2	31:0	Slot 0 R Address
M1.1	31:0	Slot 0 V Address
M1.0	31:0	Slot 0 U Address

SIMD4x2 Source Payload (Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included. Message registers given here could be a lower number if some of the address parameters are not included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR. All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M2.7	31:0	Ignored
M2.6	31:0	Ignored
M2.5	31:0	Slot 1 Source1 Specifies Source1 for slot 1. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
M2.4	31:0	Slot 1 Source0
M2.3	31:0	Ignored
M2.2	31:0	Ignored
M2.1	31:0	Slot 0 Source1
M2.0	31:0	Slot 0 Source0

SIMD4x2 Source Payload (AOP_CMPWR8B Only)

DWord	Bit	Description
M2.7	31:0	Slot 1 Source1 [63:32]
M2.6	31:0	Slot 1 Source1 [31:0]
M2.5	31:0	Slot 1 Source0 [63:32]
M2.4	31:0	Slot 1 Source0 [31:0]
M2.3	31:0	Slot 0 Source1 [63:32]
M2.2	31:0	Slot 0 Source1 [31:0]
M2.1	31:0	Slot 0 Source0 [63:32]
M2.0	31:0	Slot 0 Source0 [31:0]

SIMD4x2 Write Data Payload (Write Message Only)

The write data payload follows the address payload for write messages.

DWord	Bit	Description
M2.7	31:0	Slot 1 Alpha Specifies the value of the red channel to be written for slot 7. For *Untyped* messages: Format = 32 bits raw data. For *Typed* messages: Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
M2.6	31:0	Slot 1 Blue
M2.5	31:0	Slot 1 Green
M2.4	31:0	Slot 1 Red
M2.3	31:0	Slot 0 Alpha
M2.2	31:0	Slot 0 Blue
M2.1	31:0	Slot 0 Green
M2.0	31:0	Slot 0 Red

Writeback Message

This topic is currently under development.

SIMD8 DWORD Read

DWord	Bit	Description
W0.7	31:0	DWord[Offset7]
W0.6	31:0	DWord[Offset6]
W0.5	31:0	DWord[Offset5]
W0.4	31:0	DWord[Offset4]
W0.3	31:0	DWord[Offset3]
W0.2	31:0	DWord[Offset2]
W0.1	31:0	DWord[Offset1]
W0.0	31:0	DWord[Offset0]

SIMD8 QWORD Read

DWord	Bit	Description
W0.7 W0.6	63:0	QWord[Offset3]
W0.5 W0.4	63:0	QWord[Offset2]
W0.3 W0.2	63:0	QWord[Offset1]
W0.1 W0.0	63:0	QWord[Offset0]
W1.7 W1.6	63:0	QWord[Offset7]
W1.5 W1.4	63:0	QWord[Offset6]
W1.3 W1.2	63:0	QWord[Offset5]
W1.1 W1.0	63:0	QWord[Offset4]

SIMD16 Read

A SIMD16 writeback message consists of up to 8 destination registers. Which registers are returned is determined by the channel mask in the message descriptor. Each asserted channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3. The slots written within each destination register is determined by the execution mask on the “send” instruction.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Red:** Specifies the value of the red channel for slot 7. Format = 32 bits raw data.
W0.6	31:0	Slot 6 Red
W0.5	31:0	Slot 5 Red
W0.4	31:0	Slot 4 Red
W0.3	31:0	Slot 3 Red
W0.2	31:0	Slot 2 Red
W0.1	31:0	Slot 1 Red
W0.0	31:0	Slot 0 Red
W1.7	31:0	Slot 15 Red
W1.6	31:0	Slot 14 Red
W1.5	31:0	Slot 13 Red
W1.4	31:0	Slot 12 Red
W1.3	31:0	Slot 11 Red
W1.2	31:0	Slot 10 Red
W1.1	31:0	Slot 9 Red
W1.0	31:0	Slot 8 Red
W2		Slots 7:0 Green
W3		Slots 15:8 Green
W4		Slots 7:0 Blue
W5		Slots 15:8 Blue
W6		Slots 7:0 Alpha
W7		Slots 15:8 Alpha

SIMD8 Read

A SIMD8 writeback message consists of up to 4 destination registers. Which registers are returned is determined by the channel mask in the message descriptor. Each asserted channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0, and alpha to regid+1. The slots written within each destination register is determined by the execution mask on the "send" instruction.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Red:** Specifies the value of the red channel for slot 7. For *Untyped* messages: Format = 32 bits raw data. For *Typed* messages: Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
W0.6	31:0	Slot 6 Red
W0.5	31:0	Slot 5 Red
W0.4	31:0	Slot 4 Red
W0.3	31:0	Slot 3 Red
W0.2	31:0	Slot 2 Red
W0.1	31:0	Slot 1 Red
W0.0	31:0	Slot 0 Red
W1		Slots 7:0 Green
W2		Slots 7:0 Blue
W3		Slots 7:0 Alpha

SIMD8 Read (Tile W)

The slots written within each destination register is determined by the execution mask on the “send” instruction.

DWord	Bit	Description
M5.7	31:8	Reserved (0)
	7:0	Slot 7 Red Specifies the value of the red channel to be written for slot 7. For *Typed* messages: Format = U8
M5.6	31:8	Reserved (0)
	7:0	Slot 6 Red
M5.5	31:8	Reserved (0)
	7:0	Slot 5 Red
M5.4	31:8	Reserved (0)
	7:0	Slot 4 Red
M5.3	31:8	Reserved (0)
	7:0	Slot 3 Red
M5.2	31:8	Reserved (0)
	7:0	Slot 2 Red
M5.1	31:8	Reserved (0)
	7:0	Slot 1 Red
M5.0	31:8	Reserved (0)
	7:0	Slot 0 Red

SIMD4x2 Read

A SIMD4x2 writeback message always consists of a single message register containing all four color channels of each of the two slots. The channel mask bits as well as the execution mask on the "send" instruction are used to determine which of the channels in the destination register are overwritten. If any of the four execution mask bits for a slot is asserted, that slot is considered to be active. The active channels in the channel mask will be written in the destination register for that slot. If the slot is inactive (all four execution mask bits deasserted), none of the channels for that slot will be written in the destination register.

DWord	Bit	Description
W0.7	31:0	**Slot 1 Alpha:** Specifies the value of the pixel's alpha channel. Format = 32 bits raw data.
W0.6	31:0	Slot 1 Blue
W0.5	31:0	Slot 1 Green
W0.4	31:0	Slot 1 Red
W0.3	31:0	Slot 0 Alpha
W0.2	31:0	Slot 0 Blue
W0.1	31:0	Slot 0 Green
W0.0	31:0	Slot 0 Red

SIMD16 Atomic Operation

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Return Data:** Specifies the value of the return data for slot 7. Format = U32
W0.6	31:0	Slot 6 Return Data
W0.5	31:0	Slot 5 Return Data
W0.4	31:0	Slot 4 Return Data
W0.3	31:0	Slot 3 Return Data
W0.2	31:0	Slot 2 Return Data
W0.1	31:0	Slot 1 Return Data
W0.0	31:0	Slot 0 Return Data
W1.7	31:0	Slot 15 Return Data
W1.6	31:0	Slot 14 Return Data
W1.5	31:0	Slot 13 Return Data
W1.4	31:0	Slot 12 Return Data
W1.3	31:0	Slot 11 Return Data
W1.2	31:0	Slot 10 Return Data
W1.1	31:0	Slot 9 Return Data
W1.0	31:0	Slot 8 Return Data

SIMD16 Atomic Operation (AOP_CMPWR8B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR8B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Return Data[31:0]:** Specifies the value of the return data for slot 7. Format = U32
W0.6	31:0	Slot 6 Return Data[31:0]
W0.5	31:0	Slot 5 Return Data[31:0]
W0.4	31:0	Slot 4 Return Data[31:0]
W0.3	31:0	Slot 3 Return Data[31:0]
W0.2	31:0	Slot 2 Return Data[31:0]
W0.1	31:0	Slot 1 Return Data[31:0]
W0.0	31:0	Slot 0 Return Data[31:0]
W1.7	31:0	Slot 15 Return Data[31:0]
W1.6	31:0	Slot 14 Return Data[31:0]
W1.5	31:0	Slot 13 Return Data[31:0]
W1.4	31:0	Slot 12 Return Data[31:0]
W1.3	31:0	Slot 11 Return Data[31:0]
W1.2	31:0	Slot 10 Return Data[31:0]
W1.1	31:0	Slot 9 Return Data[31:0]
W1.0	31:0	Slot 8 Return Data[31:0]
W2		Slot 7:0 Return Data[63:32]
W3		Slot 15:8 Return Data[63:32]

SIMD8 Atomic Operation

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Return Data:** Specifies the value of the return data for slot 7. Format = U32
W0.6	31:0	Slot 6 Return Data
W0.5	31:0	Slot 5 Return Data
W0.4	31:0	Slot 4 Return Data
W0.3	31:0	Slot 3 Return Data
W0.2	31:0	Slot 2 Return Data
W0.1	31:0	Slot 1 Return Data
W0.0	31:0	Slot 0 Return Data

SIMD8 Atomic Operation (AOP_CMPWR8B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR8B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Return Data[31:0]:** Specifies the value of the return data for slot 7. Format = U32
W0.6	31:0	Slot 6 Return Data[31:0]
W0.5	31:0	Slot 5 Return Data[31:0]
W0.4	31:0	Slot 4 Return Data[31:0]
W0.3	31:0	Slot 3 Return Data[31:0]
W0.2	31:0	Slot 2 Return Data[31:0]
W0.1	31:0	Slot 1 Return Data[31:0]
W0.0	31:0	Slot 0 Return Data[31:0]
W1.7	31:0	Slot 7 Return Data[63:32]
W1.6	31:0	Slot 6 Return Data[63:32]
W1.5	31:0	Slot 5 Return Data[63:32]
W1.4	31:0	Slot 4 Return Data[63:32]
W1.3	31:0	Slot 3 Return Data[63:32]
W1.2	31:0	Slot 2 Return Data[63:32]
W1.1	31:0	Slot 1 Return Data[63:32]
W1.0	31:0	Slot 0 Return Data[63:32]

SIMD4x2 Atomic Operation

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	reserved – not written to GRF
W0.6	31:0	reserved – not written to GRF
W0.5	31:0	reserved – not written to GRF
W0.4	31:0	**Slot 1 Return Data:** Specifies the value of the return data for slot 1. Format = U32
W0.3	31:0	reserved – not written to GRF
W0.2	31:0	reserved – not written to GRF
W0.1	31:0	reserved – not written to GRF
W0.0	31:0	Slot 0 Return Data

SIMD4x2 Atomic Operation (AOP_CMPWR8B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR8B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	reserved – not written to GRF
W0.6	31:0	reserved – not written to GRF
W0.5	31:0	Slot 1 Return Data: [63:32]
W0.4	31:0	Slot 1 Return Data: [31:0]
W0.3	31:0	reserved – not written to GRF
W0.2	31:0	reserved – not written to GRF
W0.1	31:0	Slot 0 Return Data: [63:32]
W0.0	31:0	Slot 0 Return Data[31:0]

A64 Scattered Read or Write

This message takes a set of offsets, and reads or writes SIMD8 scattered Qwords/Dwords/Bytes starting at each offset. The Qword form of this message requires aligned Qword and the Dword form requires aligned Dwords.The byte form of this message takes a set of offsets, and reads or writes simd8 scattered and possibly misaligned bytes, words, dwords or qwords starting at each offset.

Programming Note	
Context:	A64 Scattered Read or Write
Writes to overlapping addresses will have undefined write ordering. Hardware does check for and optimize for cases where offsets are equal or contiguous, however for optimal performance in some these cases a different message may provide higher performance.	

Programming Note	
Context:	A64 Scattered Read or Write
Only stateless message can be used. The message header is forbidden these A64 messages, and the offsets are from a base of absolute 0	

Applications:

SIMD8 reads where the indices are different (read one channel per message)

SIMD8 writes where the indices are different (write one channel per message)

general purpose QWord/Dword/Bytes scatter/gathering, used by media

Execution Mask. The low 8 bits execution mask are used to determine which Qwords/Dwords/Bytes are read into the destination GRF register (for read), or which Qwords/Dwords/Bytes are written to the surface (for write).

Canonical address check, if post address computation, bits [63:48] don't match bit [47] then a general protect fault will occurs. If current mode is an non-faulting mode, this error will be treated as out of bounds.

Message Descriptor

Bits	Description
13	Reserved: MBZ
12	Ignored
11:10	**Data Size.** Specifies the data size for each slot: 0: 1 block 1: 2 blocks 2: 4 blocks 3: 8 blocks (not valid for QWords)
9:8	**Block Size.** Specifies the size of blocks to be read or written: 10: QWords 01: DWords 00: Bytes All other encodings are reserved.

Message Address Payload

DWord	Bit	Description
M1.7 M1.6	63:0	Offset 3. Specifies the Byte offset of QWord 3 Specifies the Byte offset of DWord 3 Specifies the Byte offset of Byte3 Format = U64 Qword Range = [0,00007FFFFFFFFF8h] or [FFFF800000000000,FFFFFFFFFFFFFF8h] Dword Range = [0,00007FFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFCh] Byte Range = [0,00007FFFFFFFFFFh] or [FFFF800000000000,FFFFFFFFFFFFFFFh]
M1.5 M1.4	63:0	Offset 2
M1.3 M1.2	63:0	Offset 1
M1.1 M1.0	63:0	Offset 0
M2.7 M2.6	63:0	Offset 7
M2.5 M2.4	63:0	Offset 6
M2.3 M2.2	63:0	Offset 5
M2.1 M2.0	63:0	Offset 4

Additional Message Payload (QWord Write)

For the write operation, either one or two additional registers (depending on the block size) of payload contain the data to be written.

The **Offsetn** referred to below is the **Offset n** in the address payload converted to units of QWords. The **QWord** array index is also in units of QWords.

DWord	Bit	Description
M3.7 M3.6	63:0	QWord[Offset3[0]
M3.5 M3.4	63:0	QWord[Offset2[0]
M3.3 M3.2	63:0	QWord[Offset1][0]
M3.1 M3.0	63:0	QWord[Offset0][0]
M4.7 M4.6	63:0	QWord[Offset7][0]
M4.5 M4.4	63:0	QWord[Offset6][0]
M4.3 M4.2	63:0	QWord[Offset5][0]
M4.1 M4.0	63:0	QWord[Offset4][0]
...	...	...
M9.7 M9.6	63:0	QWord[Offset3][3]
M9.5 M9.4	63:0	QWord[Offset2][3]

DWord	Bit	Description
M9.3 M9.2	63:0	QWord[Offset1][3]
M9.1 M9.0	63:0	QWord[Offset0][3]
M10.7 M10.6	63:0	QWord[Offset7][3]
M10.5 M10.4	63:0	QWord[Offset6][3]
M10.3 M10.2	63:0	QWord[Offset5][3]
M10.1 M10.0	63:0	QWord[Offset4][3]

Additional Message Payload (DWord Write)

For the write operation, either one to four additional registers (depending on the block size) of payload contain the data to be written.

The **Offsetn** referred to below is the **Offset n**in the address payload converted to units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M3.7	31:0	DWord[Offset7][0]
M3.6	31:0	DWord[Offset6][0]
M3.5	31:0	DWord[Offset5][0]
M3.4	31:0	DWord[Offset4][0]
M3.3	31:0	DWord[Offset3][0]
M3.2	31:0	DWord[Offset2][0]
M3.1	31:0	DWord[Offset1][0]
M3.0	31:0	DWord[Offset0][0]
M4.7	31:0	DWord[Offset7][1]
M4.6	31:0	DWord[Offset6][1]
M4.5	31:0	DWord[Offset5][1]
M4.4	31:0	DWord[Offset4][1]
M4.3	31:0	DWord[Offset3][1]
M4.2	31:0	DWord[Offset2][1]
M4.1	31:0	DWord[Offset1][1]
M4.0	31:0	DWord[Offset0][1]
...	...	...
M9.7	31:0	DWord[Offset7][6]
M9.6	31:0	DWord[Offset6][6]
M9.5	31:0	DWord[Offset5][6]
M9.4	31:0	DWord[Offset4][6]
M9.3	31:0	DWord[Offset3][6]
M9.2	31:0	DWord[Offset2][6]
M9.1	31:0	DWord[Offset1][6]
M9.0	31:0	DWord[Offset0][6]
M10.7	31:0	DWord[Offset7][7]
M10.6	31:0	DWord[Offset6][7]
M10.5	31:0	DWord[Offset5][7]
M10.4	31:0	DWord[Offset4][7]
M10.3	31:0	DWord[Offset3][7]
M10.2	31:0	DWord[Offset2][7]

DWord	Bit	Description
M10.1	31:0	DWord[Offset1][7]
M10.0	31:0	DWord[Offset0][7]

Additional Message Payload (8 Byte Write)

For the write operation, two additional registers of payload contain the data to be written.

The **Offsetn** referred to below is **Offset n** in the address payload and is in units of bytes.

DWord	Bits	Description
M3.7	31:0	Byte[Offset3][7:4]
M3.6	31:0	Byte[Offset3][3:0]
M3.5	31:0	Byte[Offset2][7:4]
M3.4	31:0	Byte[Offset2][3:0]
M3.3	31:0	Byte[Offset1][7:4]
M3.2	31:0	Byte[Offset1][3:0]
M3.1	31:0	Byte[Offset0][7:4]
M3.0	31:0	Byte[Offset0][3:0]
M4.7	31:0	Byte[Offset7][7:4]
M4.6	31:0	Byte[Offset7][3:0]
M4.5	31:0	Byte[Offset6][7:4]
M4.4	31:0	Byte[Offset6][3:0]
M4.3	31:0	Byte[Offset5][7:4]
M4.2	31:0	Byte[Offset5][3:0]
M4.1	31:0	Byte[Offset4][7:4]
M4.0	31:0	Byte[Offset4][3:0]

Additional Message Payload (1 or 2 or 4 Byte Write)

For the write operation, two additional registers of payload contain the data to be written.

The **Offsetn** referred to below is **Offset n** in the address payload and is in units of bytes. The length of **Data** written depends on the **Data Size** and is right-justified within the 32-bit field. The upper bits are ignored for 1 byte and 2 byte **Data Size**.

DWord	Bits	Description
M3.7	31:0	Byte[Offset7][3:0]
M3.6	31:0	Byte[Offset6][3:0]
M3.5	31:0	Byte[Offset5][3:0]
M3.4	31:0	Byte[Offset4][3:0]
M3.3	31:0	Byte[Offset3][3:0]
M3.2	31:0	Byte[Offset2][3:0]
M3.1	31:0	Byte[Offset1][3:0]
M3.0	31:0	Byte[Offset0][3:0]

Writeback Message (QWord Read)

For the read operation, the writeback message consists of either two to eight registers depending on the **Data size**.

The **Offsetn** referred to below is the **Offset n** in the address payload converted to units of QWords. The **QWord** array index is also in units of QWords.

DWord	Bits	Description
W0.7 W0.6	63:0	QWord[Offset3][0]
W0.5 W0.4	63:0	QWord[Offset2][0]
W0.3 W0.2	63:0	QWord[Offset1][0]
W0.1 W0.0	63:0	QWord[Offset0][0]
W1.7 W1.6	63:0	QWord[Offset7][0]

DWord	Bits	Description
W1.5 W1.4	63:0	QWord[Offset6][0]
W1.3 W1.2	63:0	QWord[Offset5][0]
W1.1 W1.0	63:0	QWord[Offset4][0]
...	...	...
W6.7 W6.6	63:0	QWord[Offset3][3]
W6.5 W6.4	63:0	QWord[Offset2][3]
W6.3 W6.2	63:0	QWord[Offset1][3]
W6.1 W6.0	63:0	QWord[Offset0][3]
W7.7 W7.6	63:0	QWord[Offset7][3]
W7.5 W7.4	63:0	QWord[Offset6][3]
W7.3 W7.2	63:0	QWord[Offset5][3]
W7.1 W7.0	63:0	QWord[Offset4][3]

Writeback Message (DWord Read)

For the read operation, the writeback message consists of either two to eight registers depending on the block size.

The **Offsetn** referred to below is the **Offset n** in the address payload converted to units of DWords. The **DWord** array index is also in units of DWords.

DWord	Bits	Description
W0.7	31:0	DWord[Offset7][0]
W0.6	31:0	DWord[Offset6][0]
W0.5	31:0	DWord[Offset5][0]
W0.4	31:0	DWord[Offset4][0]
W0.3	31:0	DWord[Offset3][0]
W0.2	31:0	DWord[Offset2][0]
W0.1	31:0	DWord[Offset1][0]
W0.0	31:0	DWord[Offset0][0]
W1.7	31:0	DWord[Offset7][1]
W1.6	31:0	DWord[Offset6][1]
W1.5	31:0	DWord[Offset5][1]
W1.4	31:0	DWord[Offset4][1]
W1.3	31:0	DWord[Offset3][1]
W1.2	31:0	DWord[Offset2][1]
W1.1	31:0	DWord[Offset1][1]
W1.0	31:0	DWord[Offset0][1]
...	...	...
W6.7	31:0	DWord[Offset7][6]
W6.6	31:0	DWord[Offset6][6]
W6.5	31:0	DWord[Offset5][6]
W6.4	31:0	DWord[Offset4][6]
W6.3	31:0	DWord[Offset3][6]
W6.2	31:0	DWord[Offset2][6]
W6.1	31:0	DWord[Offset1][6]
W6.0	31:0	DWord[Offset0][6]
W7.7	31:0	DWord[Offset7][7]
W7.6	31:0	DWord[Offset6][7]
W7.5	31:0	DWord[Offset5][7]
W7.4	31:0	DWord[Offset4][7]
W7.3	31:0	DWord[Offset3][7]
W7.2	31:0	DWord[Offset2][7]
W7.1	31:0	DWord[Offset1][7]
W7.0	31:0	DWord[Offset0][7]

Writeback Message (8 Byte Read)

For the read operation, the writeback message consists of two register.

The **Offsetn** referred to below is **Offset n** in the address payload and is in units of bytes.

DWord	Bits	Description
W0.7	31:0	Byte[Offset3][7:4]
W0.6	31:0	Byte[Offset3][3:0]
W0.5	31:0	Byte[Offset2][7:4]
W0.4	31:0	Byte[Offset2][3:0]
W0.3	31:0	Byte[Offset1][7:4]
W0.2	31:0	Byte[Offset1][3:0]
W0.1	31:0	Byte[Offset0][7:4]
W0.0	31:0	Byte[Offset0][3:0]
W1.7	31:0	Byte[Offset7][7:4]
W1.6	31:0	Byte[Offset7][3:0]
W1.5	31:0	Byte[Offset6][7:4]
W1.4	31:0	Byte[Offset6][3:0]
W1.3	31:0	Byte[Offset5][7:4]
W1.2	31:0	Byte[Offset5][3:0]
W1.1	31:0	Byte[Offset4][7:4]
W1.0	31:0	Byte[Offset4][3:0]

Writeback Message (1 or 2 or 4 Byte Read)

For the read operation, the writeback message consists of one register.

The **Offsetn** referred to below is **Offset n** in the address payload and is in units of bytes. The length of **Data** written depends on the **Data Size** and is right-justified within the 32-bit field and only the requested bytes are written to the GRF.

DWord	Bits	Description
W0.7	31:0	Byte[Offset7][3:0]
W0.6	31:0	Byte[Offset6][3:0]
W0.5	31:0	Byte[Offset5][3:0]
W0.4	31:0	Byte[Offset4][3:0]
W0.3	31:0	Byte[Offset3][3:0]
W0.2	31:0	Byte[Offset2][3:0]
W0.1	31:0	Byte[Offset1][3:0]
W0.0	31:0	Byte[Offset0][3:0]

A64 Untyped Atomic Operation

A64 Untyped Atomic Operation allows direct read/write stateless accesses.

This message supports 2 data sizes: DWORD and QWORD.

These messages use the RAW format, perform no type conversion. It uses just the U address parameter, which specifies the byte offset, which must be a multiple of 8 for QWORD and a multiple of 4 for DWORD. The atomic operation messages only access the first QWord.

The atomic operation messages cause atomic read-modify-write operations on the "destination" location addressed. In the table below, the new value of the destination (new_dst) is computed as indicated based on the old value of the destination (old_dst) and up to two sources included in the message (src0 and src1). Optionally, a value can be returned by the message (ret).

The atomic operations guarantee that the read and the write are performed atomically, meaning that no read or write to the same memory location from this thread or any other thread can occur between the read and the write.

The following atomic operations are available for QWORD data size, along with the specific operation performed for each and the return value:

Atomic Operation	new_dst	ret
AOP_AND	old_dst & src0	old_dst
AOP_OR	old_dst | src0	old_dst
AOP_XOR	old_dst ^ src0	old_dst
AOP_MOV	src0	old_dst
AOP_INC	old_dst + 1	old_dst
AOP_DEC	old_dst – 1	old_dst
AOP_ADD	old_dst + src0	old_dst
AOP_SUB	old_dst – src0	old_dst
AOP_REVSUB	src0 – old_dst	old_dst
AOP_IMAX	imax(old_dst, src0)	old_dst
AOP_IMIN	imin(old_dst, src0)	old_dst
AOP_UMAX	umax(old_dst, src0)	old_dst
AOP_UMIN	umin(old_dst, src0)	old_dst
AOP_CMPWR	(src0 == old_dst) ? src1 : old_dst	old_dst
AOP_PREDEC	old_dst – 1	new_dst
AOP_CMPWR16B	(src016B == old_dst16B) ? src116B : old_dst16B	old_dst16B

Programming Note	
Context:	A64 Untyped Atomic Operation
• src016B is 16 bytes, src116B is 16 Bytes, and old_dst16B is 16 bytes in length. • AOP_CMPWR16B is not supported for SLM. • AOP_CMPWR16B addresses must be OWORD-aligned.	

Programming Note	
Context:	A64 Untyped Atomic Operation
AOP_CMPWR16B is not supported.	

Programming Note	
Context:	A64 Untyped Atomic Operation
imax/imin assume operands are signed 64-bit integers; umax/umin assume operands are unsigned integers. All other operations treat all values as 64-bit unsigned integers. Add and subtract operations wrap without any special indication.	

The following atomic operations are available for DWORD data size, along with the specific operation performed for each and the return value:

Atomic Operation	new_dst	ret
AOP_AND	old_dst & src0	old_dst
AOP_OR	old_dst | src0	old_dst
AOP_XOR	old_dst ^ src0	old_dst
AOP_MOV	src0	old_dst
AOP_INC	old_dst + 1	old_dst
AOP_DEC	old_dst – 1	old_dst
AOP_ADD	old_dst + src0	old_dst
AOP_SUB	old_dst – src0	old_dst
AOP_REVSUB	src0 – old_dst	old_dst
AOP_IMAX	imax(old_dst, src0)	old_dst
AOP_IMIN	imin(old_dst, src0)	old_dst
AOP_UMAX	umax(old_dst, src0)	old_dst
AOP_UMIN	umin(old_dst, src0)	old_dst
AOP_CMPWR	(src0 == old_dst) ? src1 : old_dst	old_dst
AOP_PREDEC	old_dst – 1	new_dst
AOP_CMPWR8B	(src08B == old_dst8B) ? src18B : old_dst8B	old_dst8B

Programming Note	
Context:	A64 Untyped Atomic Operation
• src08B is 8 bytes, src18B is 8 Bytes, and old_dst8B is 8 bytes in length. • AOP_CMPWR8B is not supported for SLM. • AOP_CMPWR8B addresses must be QWORD-aligned.	

Programming Note	
Context:	A64 Untyped Atomic Operation
AOP_CMPWR8B is not supported for A64 SIMD4x2. Use a SIMD8 message for AOP_CMPRW8B.	

Programming Note	
Context:	A64 Untyped Atomic Operation
imax/imin assume operands are signed 32-bit integers; umax/umin assume operands are unsigned integers. All other operations treat all values as 32-bit unsigned integers. Add and subtract operations wrap without any special indication.	

Programming Note	
Context:	A64 Untyped Atomic Operation
• Only stateless messages can be used. • The stateless model support is limited to untyped messages. Furthermore, they are treated as SURFTYPE_BUFFER and **Surface Format** of RAW. • The message header is forbidden in these A64 messages, and the offsets are from a base of absolute 0.	

Execution Mask: The low 8 bits of the execution mask determine which QWords/DWords/Bytes are read into the destination GRF register (for read), or which QWords/DWords/Bytes are written to the surface (for write).

Canonical address check: If post address computation, bits [63:48] don't match bit [47] then a general protect fault occurs. If the current mode is a non-faulting mode, this error is treated as out of bounds.

Programming Note	
Context:	A64 Untyped Atomic Operation
Writes to overlapping addresses have undefined write ordering.	

A64 Untyped Atomic Operation Message Descriptor

Bits	Description
13	Return Data Control

Bits	Description
	Specifies whether return data is sent back to the thread. Format = Enable
12	Data Size This field controls the data size of the operation Format = U1 0: DWORD size 1: QWORD
11:8	Atomic Operation Type Specifies the atomic operation to be performed. 0000: 0000: AOP_CMPWR16B / AOP_CMPWR8B 0001: AOP_AND 0010: AOP_OR 0011: AOP_XOR 0100: AOP_MOV 0101: AOP_INC 0110: AOP_DEC 0111: AOP_ADD 1000: AOP_SUB 1001: AOP_REVSUB 1010: AOP_IMAX 1011: AOP_IMIN 1100: AOP_UMAX 1101: AOP_UMIN 1110: AOP_CMPWR 1111: AOP_PREDEC

Message Payload

The message payload consists of the following:

For the read messages, only an address payload is delivered.

For the write messages, an address payload is followed by the write data payload.

For the atomic operation messages, an address payload is followed by the source payload.

The number of message registers in the write data payload is determined by the number of channel mask bits that are enabled, and the number of message registers in the source payload is determined by the atomic operation.

SIMD8 Address Payload

The payload of a SIMD8 message provides address parameters to process 8 slots. The possible address parameter is U. Each parameter takes one message register. Each parameter always takes a consistent position in the input payload.

DWord	Bit	Description
M1.7 M1.6	63:0	Slot 3 U address. Specifies the Byte offset of QWord 3 Specifies the Byte offset of DWord 3 Format = U64 Qword Range = [0,00007FFFFFFFFF8h] or [FFFF800000000000,FFFFFFFFFFFFFF8h] Dword Range = [0,00007FFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFCh]
M1.5 M1.4	63:0	Slot 2 U Address
M1.3 M1.2	63:0	Slot 1 U Address
M1.1 M1.0	63:0	Slot 0 U Address
M2.7 M2.6	63:0	Slot 7 U Address
M2.5 M2.4	63:0	Slot 6 U Address
M2.3 M2.2	63:0	Slot 5 U Address
M2.1 M2.0	63:0	Slot 4 U Address

SIMD4x2 Address Payload

The payload of a SIMD4x2 message provides address parameters to process 2 slots. The possible address parameter is U. Each parameter always takes a consistent position in the input payload.

DWord	Bit	Description
M1.7 M1.6	63:0	ignored
M1.5 M1.4	63:0	Slot 1 U Address Specifies the U Address for slot 1. Format = U64 Dword Range = [0,00007FFFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFFCh]
M1.3 M1.2	63:0	ignored
M1.1 M1.0	63:0	Slot 0 U Address

SIMD8 Source Payload (QWORD Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M3.7 M3.6	63:0	Slot 3 Source0 Specifies Source0 for slot 3. Format = S63 for AOP_IMAX and AOP_IMIN, U64 for all other operations
M3.5 M3.4	63:0	Slot 2 Source0
M3.3 M3.2	63:0	Slot 1 Source0
M3.1 M3.0	63:0	Slot 0 Source0
M4.7 M4.6	63:0	Slot 7 Source0
M4.5 M4.4	63:0	Slot 6 Source0
M4.3 M4.2	63:0	Slot 5 Source0
M4.1 M4.0	63:0	Slot 4 Source0
M5		Slot3:0 Source1
M6		Slot7:4 Source1

SIMD8 Source Payload (DWORD Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M3.7	31:0	Slot 7 Source0 Specifies Source0 for slot 7. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
M3.6	31:0	Slot 6 Source0
M3.5	31:0	Slot 5 Source0
M3.4	31:0	Slot 4 Source0
M3.3	31:0	Slot 3 Source0
M3.2	31:0	Slot 2 Source0
M3.1	31:0	Slot 1 Source0
M3.0	31:0	Slot 0 Source0
M4		Slots 7:0 Source1

SIMD8 Source Payload (AOP_CMPWR16B Only)

DWord	Bit	Description
M6.7 - M6.4	31:0	Slot 1 Source0[128:0] Specifies Source0[127:0] for slot 1. Format = U128
M6.3 - M6.0	31:0	Slot 0 Source0[127:0]
M7		Slot 3:2 Source0
M8		Slot 5:4 Source0
M9		Slot 7:6 Source0
M10		Slot 1:0 Source1
M11		Slot 3:2 Source1
M12		Slot 5:4 Source1
M13		Slot 7:6 Source1

SIMD8 Source Payload (AOP_CMPWR8B Only)

DWord	Bit	Description
M6.7 M6.6		Slot 3 Source0[63:0] Specifies Source0[63:0] for slot 3. Format = U64
M6.5 M6.4		Slot 2 Source0[63:0]
M6.3 M6.2		Slot 1 Source0
M6.1 M6.0		Slot 0 Source0
M7		Slot 7:4 Source0
M8		Slot 3:0 Source1
M9		Slot 7:4 Source1

SIMD4x2 Source Payload (QWORD Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included. Message registers given here could be a lower number if some of the address parameters are not included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M3.7 M3.6	63:0	ignored
M3.5 M3.4	63:0	Slot 1 Source0 Specifies Source0 for slot 1. Format = S63 for AOP_IMAX and AOP_IMIN, U64 for all other operations
M3.3 M3.2	63:0	ignored
M3.1 M3.0	63:0	Slot 0 Source0
M4		Slot1:0 Source1

SIMD4x2 Source Payload (DWORD Atomic Operation Message Only)

The source payload follows the address payload for atomic operation messages. Depending on the atomic operation, zero, one, or two sources are required. If the source is not required, it must not be included. Message registers given here could be a lower number if some of the address parameters are not included.

The following atomic operations require no sources, thus the source payload is not delivered: AOP_INC, AOP_DEC, AOP_PREDEC

The following atomic operations require both Source0 and Source1: AOP_CMPWR

All of the remaining atomic operations require Source0 only.

DWord	Bit	Description
M3.7	31:0	ignored
M3.6	31:0	ignored
M3.5	31:0	Slot 1 Source1
M3.4	31:0	Slot 1 Source0 Specifies Source0 for slot 1. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
M3.3	31:0	ignored
M3.2	31:0	ignored
M3.1	31:0	Slot 0 Source1
M3.0	31:0	Slot0 Source0

SIMD4x2 Source Payload (AOP_CMPWR16B Only)

DWord	Bit	Description
M6.7 - M6.4	31:0	Slot 1 Source0[128:0] Specifies Source0[127:0] for slot 1. Format = U128
M6.3 - M6.0	31:0	Slot 0 Source0[127:0]
M7		Slot 1:0 Source1

SIMD4x2 Source Payload (AOP_CMPWR8B Only)

DWord	Bit	Description
M6.7 M6.6		ignored
M6.5 M6.4		Slot 1 Source0[63:0] Specifies Source0[63:0] for slot 1. Format = U64
M6.3 M6.2		ignored
M6.1 M6.0		Slot 0 Source0
M7		Slot 1:0 Source1

Writeback Message

The following subsections describe the writeback messages for A64 Untyped Atomic operations.

SIMD8 Atomic Operation (QWord)

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the “send” instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7 W0.6	63:0	**Slot 3 Return Data:** Specifies the value of the return data for slot 3. Format = U64
W0.5 W0.4	63:0	Slot 2 Return Data
W0.3 w0.2	63:0	Slot 1 Return Data
W0.1 W0.0	63:0	Slot 0 Return Data

DWord	Bit	Description
W1.7 W1.6	63:0	Slot 7 Return Data
W1.5 W1.4	63:0	Slot 6 Return Data
W1.3 W1.2	63:0	Slot 5 Return Data
W1.1 W1.0	63:0	Slot 4 Return Data

SIMD8 Atomic Operation (DWord)

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7	31:0	**Slot 7 Return Data:** Specifies the value of the return data for slot 7. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
W0.6	31:0	Slot 6 Return Data
W0.5	31:0	Slot 5 Return Data
W0.4	31:0	Slot 4 Return Data
W0.3	31:0	Slot 3 Return Data
W0.2	31:0	Slot 2 Return Data
W0.1	31:0	Slot 1 Return Data
W0.0	31:0	Slot 0 Return Data

SIMD8 Atomic Operation (AOP_CMPWR16B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR16B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7 - W0.4		**Slot 1 Return Data[127:0]:** Specifies the value of the return data for slot 1. Format = U128
W0.3 - W0.0		Slot 0 Return Data[127:0]
W1		Slot 3:2 Return Data
W2		Slot 5:4 Return Data
W3		Slot 7:6 Return Data

SIMD8 Atomic Operation (AOP_CMPWR8B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR8B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction

DWord	Bit	Description
W0.7 W0.6	63:0	**Slot 3 Return Data:** Specifies the value of the return data for slot 3. Format = U64
W0.5 W0.4	63:0	Slot 2 Return Data
W0.3 w0.2	63:0	Slot 1 Return Data
W0.1 W0.0	63:0	Slot 0 Return Data
W1.7 W1.6	63:0	Slot 7 Return Data
W1.5 W1.4	63:0	Slot 6 Return Data
W1.3 W1.2	63:0	Slot 5 Return Data
W1.1 W1.0	63:0	Slot 4 Return Data

SIMD4x2 Atomic Operation (QWord)

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7 W0.6	63:0	reserved
W0.5 W0.4	63:0	**Slot 1 Return Data:** Specifies the value of the return data for slot 1. Format = U64
W0.3 w0.2	63:0	reserved
W0.1 W0.0	63:0	Slot 0 Return Data

SIMD4x2 Atomic Operation (DWord)

A writeback message is only returned for an Atomic Operation message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W3.7	31:0	reserved
W3.6	31:0	reserved
W3.5	31:0	reserved
W3.4	31:0	**Slot 1 Return Data:** Specifies the value of the return data for slot 1. Format = S31 for AOP_IMAX and AOP_IMIN, U32 for all other operations
W3.3	31:0	reserved
W3.2	31:0	reserved
W3.1	31:0	reserved
W3.0	31:0	Slot 0 Return Data

SIMD4x2 Atomic Operation (AOP_CMPWR16B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR16B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction indicates which channels in the destination registers are overwritten.

DWord	Bit	Description
W0.7 - W0.4		**Slot 1 Return Data[127:0]:** Specifies the value of the return data for slot 1. Format = U128
W0.3 - W0.0		Slot 0 Return Data[127:0]

SIMD4x2 Atomic Operation (AOP_CMPWR8B Only)

A writeback message is only returned for an Atomic Operation AOP_CMPWR8B message if the **Send Return Data** field in the message descriptor is enabled. The execution mask on the "send" instruction

DWord	Bit	Description
W0.7 W0.6	63:0	reserved
W0.5 W0.4	63:0	**Slot 1 Return Data:** Specifies the value of the return data for slot 1. Format = U64
W0.3 w0.2	63:0	reserved
W0.1 W0.0	63:0	Slot 0 Return Data

A64 Untyped Atomic Float Add Operation

A64 Untyped Atomic Float Add Operation allows direct read/write stateless accesses.

This message supports 2 data sizes: DWORD and QWORD.

These messages use the RAW format, perform no type conversion. It uses just the U address parameter, which specifies the byte offset, which must be a multiple of 8 for QWORD and a multiple of 4 for DWORD. The atomic operation messages will only access the first QWORD.

The atomic operation messages cause atomic read-modify-write operations on the "destination" location addressed. In the table below, the new value of the destination (new_dst) is computed as indicated based on the old value of the destination (old_dst) and up to two sources included in the message (src0 and src1). Optionally, a value can be returned by the message (ret).

Programming Note	
Context:	A64 Untyped Atomic Float Add Operation
• Only stateless message can be used. • The message header is forbidden these A64 messages, and the offsets are from a base of absolute 0.	

Canonical address check: If post address computation, bits [63:48] don't match bit [47] then a General Protect fault occurs. If the current mode is a non-faulting mode, this error is treated as out of bounds.

Untyped Atomic Float Add Operation Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	Reserved
11	Data Size This field controls the data size of the operation Format = U1 0: DWORD size 1: QWORD
10:8	Reserved

Message Payload

SIMD8 Address Payload

The payload of a SIMD8 message provides address parameters to process 8 slots. The possible address parameter is U. The number of parameters required depends on the surface type being accessed. Each parameter takes one message register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

DWord	Bit	Description
M1.7 M1.6	63:0	Slot 3 U address. Specifies the Byte offset of QWord 3 Specifies the Byte offset of DWord 3 Format = U64 Qword Range = [0,00007FFFFFFFFF8h] or [FFFF800000000000,FFFFFFFFFFFFFF8h] Dword Range = [0,00007FFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFCh]
M1.5 M1.4	63:0	Slot 2 U Address
M1.3 M1.2	63:0	Slot 1 U Address
M1.1 M1.0	63:0	Slot 0 U Address
M2.7 M2.6	63:0	Slot 7 U Address
M2.5 M2.4	63:0	Slot 6 U Address
M2.3 M2.2	63:0	Slot 5 U Address
M2.1 M2.0	63:0	Slot 4 U Address

SIMD8 Source Payload (QWORD Atomic Operation Message Only)

For the write operation two additional registers (depending on the block size) of payload contain the data to be written.

The **QWord** array index is also in units of QWords.

DWord	Bit	Description
M3.7 M3.6	63:0	Slot 3 Src0
M3.5 M3.4	63:0	Slot 2 Src0
M3.3 M3.2	63:0	Slot 1 Src0
M3.1 M3.0	63:0	Slot 0 Src0
M4.7 M4.6	63:0	Slot 7 Src0
M4.5 M4.4	63:0	Slot 6 Src0
M4.3 M4.2	63:0	Slot 5 Src0
M4.1 M4.0	63:0	Slot 4 Src0

SIMD8 Source Payload (DWORD Atomic Operation Message Only)

For the write operation two additional registers (depending on the block size) of payload contain the data to be written.

The **DWord** array index is also in units of DWords.

DWord	Bit	Description
M3.7	31:0	Slot 7 Src0
M3.6	31:0	Slot 6 Src0
M3.5	31:0	Slot 5 Src0
M3.4	31:0	Slot 4 Src0
M3.3	31:0	Slot 3 Src0
M3.2	31:0	Slot 2 Src0
M3.1	31:0	Slot 1 Src0
M3.0	31:0	Slot 0 Src0

Writeback Message

A64 Untyped Surface Read or Write Operation

The A64 untyped surface message, both of which suse the RAW surface format, perform no type conversions. A raw buffer uses just the U address parameter, which specifies the byte offset, which must be a multiple of 4.

For both raw messages up to 4 DWords are accessed beginning at the byte address determined. These 4 DWords correspond to the red, green, blue, and alpha channels in that order with red mapping to the lowest order DWord.

Programming Note	
Context:	A64 Untyped Surface Read or Write Operation
• Only stateless message can be used. Furthermore, they are treated as SURFTYPE_BUFFER and **Surface Format** of RAW. • For untyped messages accessing SURFTYPE_BUFFER, the U address (byte offset) must be DWord-aligned (low 2 bits must be zero). • The message header is forbidden in these A64 messages, and the offsets are from a base of absolute 0.	

Execution Mask:

SIMD8: The low 8 bits of the execution mask are used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write).

SIMD4x2: Each group of 4 bits within the low 8 bits of the execution mask are ORed together to create two bits used to determine which slots are read into the destination GRF register.

Canonical address check: if post address computation, bits [63:48] don't match bit [47] then a general protect fault occurs. If the current mode is a non-faulting mode, this error is treated as out of bounds.

Untyped Surface Read or Write Message Descriptor

Bits	Description
13:12	**SIMD Mode** Format = U2 0: SIMD4x2 1: Reserved 2: SIMD8 3: Reserved
11	**Alpha Channel Mask** For the read message, indicates that alpha will be included in the writeback message. For the write message, indicates that alpha is included in the message payload, and that alpha will be written to the surface. 0: Alpha channel included. 1: Alpha channel not included. Programming Notes: For the *Untyped Surface Write* message, each channel mask cannot be 0 unless all of the lower mask bits are also zero. This means that the only 4-bit channel mask values allowed are 0000b, 1000b, 1100b, and 1110b. Other messages allow any combination of channel masks. For the *Untyped Surface Read* message, at least one of the channels must be unmasked (the 4-bit channel mask cannot be 1111b).
10	Blue Channel Mask
9	Green Channel Mask
8	Red Channel Mask

Message Payload

The following subsections describe the message payloads for A64 Untyped Surface Read or Write Operations.

Read or write messages use the SIMD8 Address Payload or the SIMD4x2 Address Payload.

Only write messages use the SIMD8 DWORD Write Data Payload or the SIMD4x2 DWORD Write Data Payload.

SIMD8 Address Payload

The payload of a SIMD8 message provides address parameters to process 8 slots. The possible address parameter is U. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

DWord	Bits	Description
M1.7 M1.6	63:0	Slot 3 U Address Specifies the U Address for slot 3. Format = U64 Dword Range = [0,00007FFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFFCh]
M1.5 M1.4	63:0	Slot 2 U Address
M1.3 M1.2	63:0	Slot 1 U Address
M1.1 M1.0	63:0	Slot 0 U Address
M2.7 M2.6	63:0	Slot 7 U Address
M2.5 M2.4	63:0	Slot 6 U Address
M2.3 M2.2	63:0	Slot 5 U Address
M2.1 M2.0	63:0	Slot 4 U Address

SIMD4x2 Address Payload

The payload of a SIMD4x2 message provides address parameters to process 2 slots. The possible address parameter is U. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

DWord	Bits	Description
M1.7 M1.6	63:0	Ignored
M1.5 M1.4	63:0	Slot 1 U Address Specifies the U Address for slot 1. Format = U64 Dword Range = [0,00007FFFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFFCh]
M1.3 M1.2	63:0	Ignored
M1.1 M1.0	63:0	Slot 0 U Address

SIMD8 DWORD Write Data Payload (Write Message Only)

The write data payload follows the address payload for write messages. Actual position within the message may vary if some of the parameters are not included or if some of the channel mask bits are asserted. Any parameter or write channel not included in the payload is skipped, with message phases below it being renumbered to take up the vacated space.

DWord	Bits	Description
M3.7	31:0	Slot 7 Red Specifies the value of the red channel to be written for slot 7. For *Untyped* messages: Format = 32 bits raw data.
M3.6	31:0	Slot 6 Red
M3.5	31:0	Slot 5 Red
M3.4	31:0	Slot 4 Red
M3.3	31:0	Slot 3 Red
M3.2	31:0	Slot 2 Red
M3.1	31:0	Slot 1 Red
M3.0	31:0	Slot 0 Red
M4		Slots 7:0 Green
M5		Slots 7:0 Blue
M6		Slots 7:0 Alpha

SIMD4x2 DWORD Write Data Payload (Write Message Only)

The write data payload follows the address payload for write messages.

DWord	Bits	Description
M2.7	31:0	Slot 1 Alpha Specifies the alpha channel value to write for slot 1. For *Untyped* messages: Format = 32 bits raw data.
M2.6	31:0	Slot 1 Blue
M2.5	31:0	Slot 1 Green
M2.4	31:0	Slot 1 Red
M2.3	31:0	Slot 0 Alpha
M2.2	31:0	Slot 0 Blue
M2.1	31:0	Slot 0 Green
M2.0	31:0	Slot 0 Red

Writeback Message

For A64 Untyped Surface Read or Write operations the writeback messages are SIMD8 DWORD Read and SIMD4x2 DWORD Read.

SIMD8 DWORD Read

A SIMD8 writeback message consists of up to 8 destination registers. Which registers are returned is determined by the channel mask in the message descriptor. Each asserted channel mask results in the destination register of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3. The slots written within each destination register are determined by the execution mask on the “send” instruction.

DWord	Bits	Description
W0.7	31:0	**Slot 7 Red:** Specifies the value of the red channel for slot 7. For *Untyped* messages: Format = 32 bits raw data.
W0.6	31:0	Slot 6 Red
W0.5	31:0	Slot 5 Red
W0.4	31:0	Slot 4 Red
W0.3	31:0	Slot 3 Red
W0.2	31:0	Slot 2 Red
W0.1	31:0	Slot 1 Red
W0.0	31:0	Slot 0 Red
W1		Slots 7:0 Green
W2		Slots 7:0 Blue
W3		Slots 7:0 Alpha

SIMD4x2 DWORD Read

DWord	Bits	Description
W0.7	31:0	Slot 1 Alpha Specifies the alpha channel value to write for slot 1. For *Untyped* messages: Format = 32 bits raw data.
W0.6	31:0	Slot 1 Blue
W0.5	31:0	Slot 1 Green
W0.4	31:0	Slot 1 Red
W0.3	31:0	Slot 0 Alpha
W0.2	31:0	Slot 0 Blue
W0.1	31:0	Slot 0 Green
W0.0	31:0	Slot 0 Red

A64 Block Read or Write Operation

Programming Note	
Context:	A64 Block Read or Write Operation
• OWord & Dual Oword offsets must be aligned on OWord boundaries. • Unaligned OWord offsets must be aligned on DWord boundaries. • HWord offsets must be aligned on HWord boundaries.	

Execution Mask (OWord read/write). The low 8 bits of the execution mask are used to enable the 8 channels in the first and third GRF registers returned (W0, W2) for read, or the first and third write registers sent (M1, M3). The high 8 bits are used similarly for the second and fourth (W1, W3 or M2, M4). For reads, any mask bit asserted within a group of four will cause the entire OWord to be read and returned to the destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

For the 1-OWord messages, only the low 8 bits of the execution mask are used. Either the low 4 bits or the high 4 bits, depending on the position of the OWord to be read or written, is used as the single group of four with behavior following that in the preceding paragraph.

The above behavior enables a SIMD16 thread to use the 8-OWord form of this message to access two channels (red and green) of a single scratch register across 16 pixels. A second message would access the other two channels (blue and alpha). The execution mask is used to ensure that data associated with inactive pixels are not overwritten.

Execution Mask (Dual Oword read/write). The low 8 bits of the execution mask are used to enable the 8 channels in the GRF registers returned for read, or each of the write registers sent. For reads, any mask bit asserted within a group of four causes the entire OWord to be read and returned to the

destination GRF register. For writes, each mask bit is considered for its corresponding DWord written to the destination surface.

Execution Mask (Unaligned OWord read/write). The execution mask is ignored by this message sub type.

Execution Mask (HWord read/write). The low 8 bits of the execution mask are used to enable the 8 channels in the first and third GRF registers returned (W0, W2) for read, or the first and third write registers sent (M1, M3). The high 8 bits are used similarly for the second and fourth (W1, W3 or M2, M4). For reads either any mask bit asserted within a group of four will cause the entire OWord to be read and returned to the destination GRF register or the mask bit control corresponding DWORD based on the **HWord Read/Write Channel Mode**. For writes, either any mask bit asserted within a group of four will cause the entire OWord to be written or each mask bit is considered for its corresponding DWord written to the destination surface **HWord Read/Write Channel Mode**.

For the 1-OWord messages, only the low 8 bits of the execution mask are used. Either the low 4 bits or the high 4 bits, depending on the position of the OWord to be read or written, is used as the single group of four with behavior following that in the preceding paragraph.

Canonical address check: if post address computation, bits [63:48] don't match bit [47] then a general protect fault occurs. If current mode is a non-faulting mode, this error is treated as out of bounds.

Message Descriptor

Bits	Description
13	Reserved: MBZ
12:11	Message sub-type: 00: OWord Block Read/Write 01: Unaligned OWord Block Read 10: OWord Dual Block Read/Write 11: HWord Block Read/Write
10:8	**Block Size.** Specifies the number of elements transferred. See table below.

	00	01	10	11
Block Size	Oword	UnAligned Oword	Oword Dual	Hword
000	1 OWord, read into or written from the low 128 bits of the destination register	1 OWord, read into or written from the low 128 bits of the destination register	reserved	reserved
001	1 OWord, read into or written from the high 128 bits of the destination register	1 OWord, read into or written from the high 128 bits of the destination register	1 Oword	1 HWord
010	2 OWords	2 OWords	reserved	2 HWord
011	4 OWords	4 OWords	4 OWords	4 HWord
100	8 OWords	8 OWords	reserved	8 HWord
101	reserved	reserved	reserved	reserved
110	reserved	reserved	reserved	reserved
111	reserved	reserved	reserved	reserved

Message Header

DWord	Bits	Description
M0.5	31	**HWord Read/Write Channel Mode.** This field is only used for HWord read/write messages. **0: OWord** – Channel enables in effect at the time of 'send' are interpreted such that if one or more are enabled, the read or write operation occurs on all four DWords. **1: DWord** – Channel enables in effect at the time of the 'send' are used as DWord enables, causing the read or write operation to occur only on the DWords where the corresponding channel enable is set.
	30:0	Ignored
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3 M0.2		**Block Offset 1.** Specifies the Byte offset of OWord Block 1 for OWord Dual reads. Format = U64 Dual OWord Range = [0,00007FFFFFFFFFF0h] or [FFFF800000000000,FFFFFFFFFFFFFFF0h]
M0.1 M0.0		**Block Offset 0.** Specifies the Byte offset of Block 0. Format = U64 Unaliged OWord Range = [0,00007FFFFFFFFFFCh] or [FFFF800000000000,FFFFFFFFFFFFFFFCh] Dual OWord Range = [0,00007FFFFFFFFFF0h] or [FFFF800000000000,FFFFFFFFFFFFFFF0h] OWord Range = [0,00007FFFFFFFFFF0h] or [FFFF800000000000,FFFFFFFFFFFFFFF0h] HWord Range = [0,00007FFFFFFFFFE0h] or [FFFF800000000000,FFFFFFFFFFFFFFE0h]

Message Payload (OWord Write)

For the write operation, the message payload consists of one, two, or four registers (not including the header) depending on the **Block Size** specified in the message. For the one-constant case, data is taken from either the high or low half of the payload register depending on the half selected in **Block Size**. In this case, the other half of the payload register is ignored.

DWord	Bits	Description
M1.7:4	127:0	**OWord[Offset + 1].** If the block size is 1 OWord to be written from the high 128 bits of the destination, OWord[Offset] appears in this location.
M1.3:0	127:0	OWord[Offset]
M2.7:4	127:0	OWord[Offset+3]
M2.3:0	127:0	OWord[Offset+2]
M3.7:4	127:0	OWord[Offset+5]
M3.3:0	127:0	OWord[Offset+4]
M4.7:4	127:0	OWord[Offset+7]
M4.3:0	127:0	OWord[Offset+6]

Writeback Message (OWord Read)

For the read operation, the writeback message consists of one, two, three, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

DWord	Bits	Description
W0.7:4	127:0	**OWord[Offset+1].** If the block size is 1 OWord to be loaded into the high 128 bits of the destination, OWord[Offset] appears in this location.
W0.3:0	127:0	OWord[Offset]
W1.7:4	127:0	OWord[Offset+3]
W1.3:0	127:0	OWord[Offset+2]
W2.7:4	127:0	OWord[Offset+5]
W2.3:0	127:0	OWord[Offset+4]
W3.7:4	127:0	OWord[Offset+7]
W3.3:0	127:0	OWord[Offset+6]

Writeback Message (Unaligned OWord Read)

For the read operation, the writeback message consists of one, two, or four registers depending on the **Block Size** specified in the message. For the one-constant case, data is placed in either the high or low half of the returned register depending on the half selected in **Block Size**. In this case, the other half of the register is not changed.

DWord	Bits	Description
W0.7:4	127:0	**OWord1 = *(&OWord0 + 1).** If the block size is 1 OWord to be loaded into the high 128 bits of the destination, OWord0 appears in this location.
W0.3:0	127:0	OWord0 = *Offset
W1.7:4	127:0	OWord3 = *(&OWord2 + 1)
W1.3:0	127:0	OWord2 = *(&OWord1 + 1)
W2.7:4	127:0	OWord5 = *(&OWord4 + 1)
W2.3:0	127:0	OWord4 = *(&OWord3 + 1)
W3.7:4	127:0	OWord7 = *(&OWord6 + 1)
W3.3:0	127:0	OWord6 = *(&OWord5 + 1)

Message Payload (Dual OWord Write)

For the write operation, the message payload consists of one or four registers (not including the header or the first part of the payload) depending on the **Block Size** specified in the message.

DWord	Bits	Description
M2.7:4	127:0	OWord[Offset1]
M2.3:0	127:0	OWord[Offset0]
M3.7:4	127:0	OWord[Offset1+1]
M3.3:0	127:0	OWord[Offset0+1]
M4.7:4	127:0	OWord[Offset1+2]
M4.3:0	127:0	OWord[Offset0+2]
M4.7:4	127:0	OWord[Offset1+3]
M4.3:0	127:0	OWord[Offset0+3]

Writeback Message (Dual Oword Read)

For the read operation, the writeback message consists of one or four registers depending on the **Block Size** specified in the message.

DWord	Bits	Description
W0.7:4	127:0	OWord[Offset1]
W0.3:0	127:0	OWord[Offset0]
W1.7:4	127:0	OWord[Offset1+1]
W1.3:0	127:0	OWord[Offset0+1]
W2.7:4	127:0	OWord[Offset1+2]
W2.3:0	127:0	OWord[Offset0+2]
W3.7:4	127:0	OWord[Offset1+3]
W3.3:0	127:0	OWord[Offset0+3]

Message Payload (HWord Write)

The table below illustrates the write payload for a message of block size = 4.

DWord	Bits	Description
M1.7:0	255:0	HWord[Offset]
M2.7:0	255:0	HWord[Offset+1]
M3.7:0	255:0	HWord[Offset+2]
M3.7:0	255:0	HWord[Offset+3]

Writeback Message (HWord Read)

The table below illustrates an example where 4 HWords are read through a scratch block read.

DWord	Bits	Description
W0.7:0	255:0	HWord[Offset]
W1.7:0	255:0	HWord[Offset+1]
W2.7:0	255:0	HWord[Offset+2]
W3.7:0	255:0	HWord[Offset+3]

Untyped Atomic Float Add Operation

Untyped Atomic Operation allows direct read/write accesses to surfaces. The categories of surface:

Raw buffer (untyped). These surfaces are of type SURFTYPE_BUFFER and have a surface format of RAW and a surface pitch of 1 byte. Supported via the data cache data port.

Structured buffer (untyped). These surfaces are of type SURFTYPE_STRBUF and have a surface format of RAW. Supported via the data cache data port.

This message supports 2 data sizes: Single Precision (DWORD) and Double Precision (QWORD).

The untyped surface categories, both of which use the RAW surface format, perform no type conversion. A raw buffer uses just the U address parameter, which specifies the byte offset into the surface, which must be a multiple of 8 for QWORD and a multiple of 4 for DWORD. For structured buffers, the U parameter is the index into an array of structures and the V parameter is the actual byte offset into the structure instance at index U, which must be DWord/QWord aligned as the case may be. The atomic operation messages only access the first QWord or DWord.

The atomic operation messages cause atomic read-modify-write operations on the destination location addressed. The new value of the destination (new_dest) is computed based on the old value of the destination (old_dest) by adding to it the source value (src0) included in the message. Optionally, the old value can be returned by the message (ret).

The atomic operations guarantee that the read and the write are performed atomically, meaning that no read or write to the same memory location from this thread or any other thread can occur between the read and the write.

Execution Mask:

SIMD16: The 16 bits of the execution mask are ANDed with the 16 bits of the **Pixel/Sample Mask** from the message header and the resulting mask is used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write). If the header is not present, only the execution mask is used.

SIMD8: The low 8 bits of the execution mask are ANDed with 8 bits of the **Pixel/Sample Mask** from the message header. The resulting mask is used to determine which slots are read into the destination GRF register (for read), or which slots are written to the surface (for write). If the header is not present, only the low 8 bits of the execution mask are used.

Out–of–Bounds Accesses: Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and will not modify memory contents.

Programming Note	
Context:	Untyped Atomic Float Add Operation
• Writes to overlapping addresses will have undefined write ordering. • The stateless model support is restricted to SURFTYPE_BUFFER and Surface Format of RAW. The bounds checking for the stateless message is 4GB overflow and < General State upper bound.	

Untyped Atomic Float Add Operation Message Descriptor

Bit	Description
13	Return Data Control Specifies whether return data is sent back to the thread. Format = Enable
12	SIMD Mode Format = U1 0: SIMD16 1: SIMD8
11	Data Size This field controls the data size of the operation Format = U1 0: DWORD size 1: QWORD
10:8	Reserved

Message Header

The message header for the untyped messages only needs to be delivered for pixel shader threads, where the execution mask may indicate pixels/samples that are enabled only due to derivative (LOD) calculations, but the corresponding slot on the surface must not be accessed.

DWord	Bit	Description
M0.7	31:16	Ignored
	15:0	**Pixel/Sample Mask.** This field contains the 16-bit pixel/sample mask to be used for SIMD16 and SIMD8 messages. All 16 bits are used for SIMD16 messages. For untyped SIMD8 messages, the low 8 bits of this field are used. If the header is not delivered, this field defaults to all ones. The field is ignored for SIMD4x2 messages.
M0.6	31:0	Ignored
M0.5	31:0	**Immediate Buffer Base Address. [CHV, BSW]** Specifies the surface base address for messages in which the Binding Table Index is 255 (stateless model), otherwise this field is ignored. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Message Payload

SIMD16 Address Payload

The payload of a SIMD16 message provides address parameters to process 16 slots. The possible address parameters are U and V (since SIMD16 is only supported with untyped messages). The number of parameters required depends on the surface type being accessed. Each parameter takes two message registers. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

Programming Note	
Context:	SIMD16 Address Payload
For untyped messages of surface type SURFTYPE_BUFFER, either U only can be sent or U and V can be sent. If V is sent, it is ignored.	

DWord	Bit	Description
M1.7	31:0	Slot 7 U Address Specifies the U Address for slot 7. Format = U32
M1.6	31:0	Slot 6 U Address
M1.5	31:0	Slot 5 U Address
M1.4	31:0	Slot 4 U Address
M1.3	31:0	Slot 3 U Address
M1.2	31:0	Slot 2 U Address
M1.1	31:0	Slot 1 U Address
M1.0	31:0	Slot 0 U Address
M2.7	31:0	Slot 15 U Address
M2.6	31:0	Slot 14 U Address
M2.5	31:0	Slot 13 U Address
M2.4	31:0	Slot 12 U Address
M2.3	31:0	Slot 11 U Address
M2.2	31:0	Slot 10 U Address
M2.1	31:0	Slot 9 U Address
M2.0	31:0	Slot 8 U Address
M3		Slots 7:0 V Address
M4		Slots 15:8 V Address

SIMD8 Address Payload

The payload of a SIMD8 message provides address parameters to process 8 slots. The possible address parameters are U, V. The number of parameters required depends on the surface type being accessed. Each parameter takes one message register. Each parameter always takes a consistent position in the input payload. The length field can be used to send a shorter message, but intermediate parameters cannot be skipped as there is no way to signal this.

Programming Note	
Context:	SIMD8 Address Payload
• For untyped messages of surface type SURFTYPE_BUFFER, either U only can be sent or U and V can be sent. If V is sent it is ignored. • For untyped messages of surface type SURFTYPE_STRBUF, both U and V must be sent.	

DWord	Bit	Description
M1.7	31:0	Slot 7 U Address Specifies the U Address for slot 7. Format = U32
M1.6	31:0	Slot 6 U Address
M1.5	31:0	Slot 5 U Address
M1.4	31:0	Slot 4 U Address
M1.3	31:0	Slot 3 U Address
M1.2	31:0	Slot 2 U Address
M1.1	31:0	Slot 1 U Address
M1.0	31:0	Slot 0 U Address
M2		Slots 7:0 V Address

SIMD16/SIMD8 DWORD Source Payload

Either one or two additional registers (depending on the SIMD mode) of payload contain the sources to be used.

DWord	Bit	Description
M3.7	31:0	DWord[slot7]
M3.6	31:0	DWord[slot6]
M3.5	31:0	DWord[slot5]
M3.4	31:0	DWord[slot4]
M3.3	31:0	DWord[slot3]
M3.2	31:0	DWord[slot2]
M3.1	31:0	DWord[slot1]

DWord	Bit	Description
M3.0	31:0	DWord[slot0]
M4.7	31:0	**DWord[slot15].** This message register is included only for SIMD16.
M4.6	31:0	DWord[slot14]
M4.5	31:0	DWord[slot13]
M4.4	31:0	DWord[slot12]
M4.3	31:0	DWord[slot11]
M4.2	31:0	DWord[slot10]
M4.1	31:0	DWord[slot9]
M4.0	31:0	DWord[slot8]

SIMD16/SIMD8 QWORD Source Payload

Either two or four additional registers (depending on the SIMD mode) of payload contain the sources to use.

DWord	Bits	Description
M3.7 M3.6	63:0	QWord[slot3]
M3.5 M3.4	63:0	QWord[slot2]
M3.3 M3.2	63:0	QWord[slot1]
M3.1 M3.0	63:0	QWord[slot0]
M4.7 M4.6	63:0	QWord[slot7]
M4.5 M4.4	63:0	QWord[slot6]
M4.3 M4.2	63:0	QWord[slot5]
M4.1 M4.0	63:0	QWord[slot4]
M5		**Qword[slot11:slot8].** This register is only included for SIMD16.
M6		**Qword[slot15:slot12].** This register is only included for SIMD16.

Writeback Message

SIMD16 SIMD8 DWORD Atomic Float Add

DWord	Bit	Description
W0.7	31:0	DWord[slot7]
W0.6	31:0	DWord[slot6]
W0.5	31:0	DWord[slot5]
W0.4	31:0	DWord[slot4]
W0.3	31:0	DWord[slot3]
W0.2	31:0	DWord[slot2]
W0.1	31:0	DWord[slot1]
W0.0	31:0	DWord[slot0]
W1.7	31:0	**DWord[slot15].** This message register is included only for SIMD16.
W1.6	31:0	DWord[slot14]
W1.5	31:0	DWord[slot13]
W1.4	31:0	DWord[slot12]
W1.3	31:0	DWord[slot11]
W1.2	31:0	DWord[slot10]
W1.1	31:0	DWord[slot9]
W1.0	31:0	DWord[slot8]

SIMD16/SIMD8 QWORD Atomic Float Add

DWord	Bit	Description
W0.7 W0.6	63:0	QWord[slot3]
W0.5 W0.4	63:0	QWord[slot2]
W0.3 W0.2	63:0	QWord[slot1]
W0.1 W0.0	63:0	QWord[slot0]
W1.7 W1.6	63:0	QWord[slot7]
W1.5 W1.4	63:0	QWord[slot6]
W1.3 W1.2	63:0	QWord[slot5]
W1.1	63:0	QWord[slot4]

DWord	Bit	Description
W1.0		
W2		**Qword[slot11:Offset8].** This register is only included for SIMD16.
W3		**Qword[slot15:Offset12].** This register is only included for SIMD16.

Read Surface Info

This message is used to determine information about a surface.

Message Descriptor

Bits	Description
13:8	Reserved

Address Payload

DWord	Bits	Description
M1.7	31:0	Reserved
M1.6	31:0	Reserved
M1.5	31:0	Reserved
M1.4	31:0	Reserved
M1.3	31:0	LOD Address
M1.2	31:0	R Address
M1.1	31:0	V Address
M1.0	31:0	U Address

Writeback Message

DWord	Bits	Description						
W0.7	31:0	Instruction Base Address [63:32]	Instruction Base Address [63:32]	Instruction Base Address [63:32]	Instruction Base Address [63:32]	Instruction Base Address [63:32]	Instruction Base Address [63:32]	Instruction Base Address [63:32]
W0.6	31:0	Instruction Base Address [31:0]	Instruction Base Address [31:0]	Instruction Base Address [31:0]	Instruction Base Address [31:0]	Instruction Base Address [31:0]	Instruction Base Address [31:0]	Instruction Base Address [31:0]
W0.5	31:0							
W0.4	31:0							

DWord	Bits	Description						
W0.3	31:0							
W0.2	31:0							
W0.1	31:0							
W0.0	31:0							
W1.7	31:0							
W1.6	31:0							
W1.5	31:0	0	Surface Format	Surface Format	Surface Format	Surface Format	Surface Format	Surface Format
W1.4	31:0	SURFTYPE_NULL (7h)	SURFTYPE_STRBUF (5h)	SURFTYPE_BUFFER(4h)	SURFTYPE_CUBE (3h)	SURFTYPE_3D (2h)	SURFTYPE_2D (1h)	SURFTYPE_1D (0h)
W1.3	31:0	0	0	0	MIPCount	MIPCount	MIPCount	MIPCount
W1.2	31:0	0	0	0	Surface Array ? Depth+1 : 0	(Depth+1)>>LOD	Surface Array ? Depth+1 : 0	0
W1.1	31:0	0	(Height+1)>>LOD	0	(Height+1)>>LOD	(Height+1)>>LOD	(Height+1)>>LOD	Surface Array ? Depth+1 : 0
W1.0	31:0	0	(Width+1)>>LOD	(Width+1)>>LOD	(Width+1)>>LOD	(Width+1)>>LOD	(Width+1)>>LOD	(Width+1)>>LOD

The 64-bit Instruction Base Address is specified as a 48-bit state base address and is extended to 64 bits in HW. It is reflected here so SW can read it for conversion of 64-bit instruction pointers.

LOD Information
The LOD is in-bounds if LOD < MIPCount and if MinLOD + LOD < 15. If LOD is not in-bounds then 0 is returned for the width, height, and depth values.

Transpose Read

This message enables a rectangular block of DWords to be read from the source surface and written into the GRF.

Programming Note	
Context:	Transpose Read

Accesses are allowed to SURFTYPE_NULL, reads return 0 and writes are ignored.

- The only surface type allowed is non-arrayed, non-mipmapped SURFTYPE_2D. Because of this, the stateless surface model is not supported with this message.
- The raw data from the surface is returned to the thread without any format conversion nor filtering operation.
- The target cache must be the data cache.
- The surface base address must be 32-byte aligned.
- The surface width must a multiple of DWORDs.
- The only tiling modes supported are: Tile Y and Linear.
- The only tiling modes supported are: Tile Y and Linear.
- The block width must be 1, 2, 4, or 8 DWORDs.
- The X Offset must be a multiple of the block width in bytes.
- The surface format must be 32 bpp (DWORD).
- Vertical stride, offset and boundary clamping modes from surface state are supported.
- The block height must be 1, 2, 4, or 8 rows.
- The Y Offset must be a multiple of the block height.
- Pitch must be a multiple of 64 bytes when the surface is linear.

Applications:

Block reads/writes for media.

Execution Mask. The execution mask on the send instruction for this type of message is ignored. The data that is read is determined completely by the block parameters.

Out-of-Bounds Accesses. Reads outside of the surface results in the address being clamped to the nearest edge of the surface and the pixel in the position being returned.

Message Descriptor

Bits	Description
13:8	Reserved

Message Header

DWord	Bits	Description
M0.5	31:8	Ignored
	7:0	**FFTID.** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:22	Ignored
	21:20	**Block Height.** Height in rows of block being accessed. Value / Block Height (Rows): 0 / 1; 1 / 2; 2 / 4; 3 / 8 Format = U2 Range = [0,3] representing 1 to 8 rows
	19:2	Ignored
	1:0	**Block Width.** Width in DWORDs of block being accessed. Value / Block Width (DWORDs): 0 / 1; 1 / 2; 2 / 4; 3 / 8 Format = U2 Range = [0,3] representing 1 to 8 DWORDs
M0.1	31:0	**Y offset.** The Y offset (in rows) of the upper left corner of the block into the surface.

DWord	Bits	Description
		Format = S31 Programming Note: This field must be a multiple of the block height.
M0.0	31:0	**X offset.** The X offset (in bytes) of the upper left corner of the block into the surface. Format = S31 Programming Notes: This field must be a multiple of Block Width. Must be DWord aligned.

Writeback Message

DWord	Bits	Description
W0.7	31:0	DWord[XOffset+0,YOffset+7]
W0.6	31:0	DWord[XOffset+0,YOffset+6]
W0.5	31:0	DWord[XOffset+0,YOffset+5]
W0.4	31:0	DWord[XOffset+0,YOffset+4]
W0.3	31:0	DWord[XOffset+0,YOffset+3]
W0.2	31:0	DWord[XOffset+0,YOffset+2]
W0.1	31:0	DWord[XOffset+0,YOffset+1]
W0.0	31:0	DWord[XOffset+0,YOffset+0]
W1.7	31:0	DWord[XOffset+1,YOffset+7]
W1.6	31:0	DWord[XOffset+1,YOffset+6]
W1.5	31:0	DWord[XOffset+1,YOffset+5]
W1.4	31:0	DWord[XOffset+1,YOffset+4]
W1.3	31:0	DWord[XOffset+1,YOffset+3]
W1.2	31:0	DWord[XOffset+1,YOffset+2]
W1.1	31:0	DWord[XOffset+1,YOffset+1]
W1.0	31:0	DWord[XOffset+1,YOffset+0]
...	...	...
W7.7	31:0	DWord[XOffset+7,YOffset+7]
W7.6	31:0	DWord[XOffset+7,YOffset+6]
W7.5	31:0	DWord[XOffset+7,YOffset+5]
W7.4	31:0	DWord[XOffset+7,YOffset+4]
W7.3	31:0	DWord[XOffset+7,YOffset+3]
W7.2	31:0	DWord[XOffset+7,YOffset+2]
W7.1	31:0	DWord[XOffset+7,YOffset+1]
W7.0	31:0	DWord[XOffset+7,YOffset+0]

Scratch Block Read or Write

This message performs a read or write operation of between 1 and 4 SIMD8 registers to an HWord aligned offset to scratch memory. The HWord offset into the scratch memory is provided in the message descriptor, allowing a single instruction read|write block operation in a single source instruction. 12 bits are provided for the HWord offset, allowing addressing of 4K Hword locations (128KB).

Two modes of channel-enable interpretation are provided: DWord, which support a SIMD8 or SIMD16 DWord channel-serial view of a register, and OWord, which supports a SIMD4x2 view of a register. For

operations using SIMD32 processing, two messages should be used, with one of them indicating 'H2' to select the upper 16b of the execution mask.

This message type can only be used with stateless model memory access. Thus binding table entry 0xFF is hard-coded into the execution of this message.

Applications: Scratch space reads/writes for register spill/fill operations.

Execution Mask. The low 8 bits of the execution mask are used to enable the 8 channels in the first and third GRF registers returned (W0, W2) for read, or the first and third write registers sent (M1, M3). The high 8 bits are used similarly for the second and fourth registers (W1, W3 or M2, M4).

For DWord mode, the execution mask delivered with the message dictates DWord-based control of read or write operations. For OWord mode, any one or more asserted bits within the OWord's corresponding execution mask nibble causes read or write operations to occur across all four DWords of the OWord regardless of the setting of any particular DWord's bit.

Out-of-Bounds Accesses. Reads to areas outside of the surface return 0. Writes to areas outside of the surface are dropped and do not modify memory contents.

Programming Note	
Context:	Scratch Block Read or Write
Out-of-Bounds check is disabled.	

Message Descriptor

Bits	Description
17	Operation Type: 0 = Read, 1 = Write
16	**Channel Mode:** **0: OWord** – Channel enables in effect at the time of 'send' are interpreted such that, if one or more are enabled, the read or write operation occurs on all four DWords. **1: DWord** – Channel enables in effect at the time of the 'send' are used as DWord enables, causing the read or write operation to occur only on the DWords whose corresponding channel enable is set.
15	Reserved: MBZ.
14	Reserved: MBZ
13:12	**Block Size.** Indicates the number of SIMD8 registers to be read or written: 00: 1 register 01: 2 registers 10: 4 registers 11: 8 registers
11:0	**Offset.** A 12-bit HWord offset into the memory Immediate Memory buffer as specified by binding table 0xFF.

Message Header

DWord	Bits	Description
M0.7	31:16	Ignored
	15:0	Ignored
M0.6	31:0	Ignored
M0.5	31:0	**Immediate Buffer Base Address.** Specifies the surface base address for messages in which the Binding Table Index is 255 (stateless model); otherwise this field is ignored. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:10]
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Message Payload (Write)

The table below illustrates the write payload for a message of block size = 4.

DWord	Bits	Description
M1.7:0	255:0	HWord[Offset]
M2.7:0	255:0	HWord[Offset+1]
M3.7:0	255:0	HWord[Offset+2]
M3.7:0	255:0	HWord[Offset+3]

Message Payload (Read)

Read only requires a message header and has no message address payload.

Writeback Message (Read)

The table below illustrates an example where 4 HWords are read through a scratch block read.

DWord	Bits	Description
W0.7:0	255:0	HWord[Offset]
W1.7:0	255:0	HWord[Offset+1]
W2.7:0	255:0	HWord[Offset+2]
W3.7:0	255:0	HWord[Offset+3]

Memory Fence

A memory fence message issued by a thread causes further messages issued by the thread to be blocked until all previous messages issued by the thread to that data port (data cache or render cache) have been globally observed from the point of view of other threads in the system. This includes both read and write messages.

Data is called globally observable by other threads in the system when the data values written to the memory or data cache will now be returned by other threads' read messages when using that same data port. To read globally observable data that was written to a different data port, the thread issuing the data port read message needs to flush its cache (using a memory fence or pipe control) after the program knows that the writing thread issued the memory fence that ensured the global observability.

The memory fence message has an optional commit writeback message. The commit is sent only after all previous messages by this thread to that data port have been globally observed. This writeback can be used by threads to ensure that a fence is honored across both data ports, as each data port's memory fence only honors the corresponding data port messages.

Notes
CHV, BSW supports page faulting on some data cache operations. In the event of a page fault condition, the global observability of other data port operations may impacted. For a thread to ensure that typed or untyped UAV accesses are visible to other threads, the memory fence message is used with the **Commit Enable** control to ensure that all page fault conditions in this thread have been handled. When the writeback message is returned to the thread, then any page fault conditions have been handled and the memory is globally observable to other threads. The normal usage is to issue a memory fence, source the writeback register, and then issue a gateway barrier message to release other threads to use the memory data. By using the writeback register from the fence message before issuing the next memory operation, the program guarantees that the fence has completed before the next data port message is issued.

Programming Note	
Context:	Memory Fence
The writeback message returned with **Commit Enable** does not properly ensure that all page fault conditions in this thread have been handled. The workaround is to replace the memory fence message commit enable with the following two messages in this sequence: • Memory Fence message without Commit Enable • Read Surface Info message (which returns data in a writeback message) After the Read Surface Info writeback message is received by the thread, then all page fault conditions in this thread have been handled.	

Message Descriptor

Bits	Description
13	**Commit Enable** Specifies whether the commit is returned to the thread after the fence has been honored. Format = Enable
12	**L3_Flush_RW_Data** If enabled causes the L3 to flush any RW data. If disabled RW data is not flushed.
11	**L3_Flush_Constant_Data** If enabled causes the L3 to flush any Constant data. If disabled Constant data is not flushed.
10	**L3_Flush_Texture_Data** If enabled causes the L3 to flush any Texture data. If disabled Texture data is not flushed.
9	**L3_Flush_Instructions** If enabled causes the L3 to flush any Instructions. If disabled Instructions are not flushed.
8	**Reserved:** MBZ

Programming Note	
Context:	Message Descriptor
Only one of the Flush controls (bits 12:9) may be specified per message.	

Programming Note	
Context:	Message Descriptor
The Flush controls (bits 12:9) are ignored.	

Programming Note	
Context:	Message Descriptor
The L3 has a few different partitioning schemes. In some cases RW data, Constant data and/or texture data can be mixed in the same partition. If a flush is needed for any data type in the partition then the entire partition is flushed.	

Programming Note	
Context:	Message Descriptor
The flushing of L3 is normally not needed. It is provided primarily to support workarounds, if needed.	

Programming Note	
Context:	Message Descriptor
SFID_DP_DC0 & SFID_DP_DC1 (unless forced non-coherent) are normally IA-coherent. Therefore no additional actions are needed by SW to ensure coherence. However, SW must use the memory fence (descriptor bits [12:8] can be all 0) to ensure that all memory cycles are visible throughout the memory hierarchy.	

Message Header

The fence messages consist of a single phase, and is completely ignored. The message length is always one.

DWord	Bits	Description
M0.7:0	31:0	Ignored

Writeback Message

The writeback message is only sent if **Commit Enable** in the message descriptor is set. The destination register is not modified. Memory fence messages without **Commit Enable** set do not return anything to the thread (response length is 0 and destination register is null).

DWord	Bits	Description
W0.7:0		Reserved

Pixel Data Port

Cache Agents

The data port allows access to memory via various caches. The choice of which cache to use for a given application is dictated by its restrictions, coherency issues, and how heavily that cache is used for other purposes.

The cache to use is selected by the shared function accessed.

Accessing Render Targets

Render targets are the surfaces that the final results of pixel shaders are written to. The render targets support a large set of surface formats (refer to surface formats table in *Sampling Engine* for details) with hardware conversion from the format delivered by the thread. The render target message also causes numerous side effects, including potentially alpha test, depth test, stencil test, alpha blend (which normally causes a read of the render target), and other functions. These functions are covered in the *Windower* chapter as some of them (depth/stencil test) are also partially done in the Windower.

The render target write messages are specifically for the use of pixel shader threads that are spawned by the windower, and may not be used by any other threads. This is due to the pixel scoreboard side-effects that sending of this message entails. The pixel scoreboard ensures that incorrect ordering of reads and writes to the same pixel does not occur.

Half Precision Render Target Messages

In addition to 32b components (or channels A, R, G and B), 16b components are supported via Message Specific Descriptor. These messages are referred to as Half Precision Render Target messages. Pyloads for half precision render target messages are described in the respective sections.

Programming Note	
Context:	Accessing Render Targets
Half Precision Render Target Write messages do not support UNIT formats.	

Message Sequencing Summary

This section summarizes the sequencing that occurs for each legal render target write message. All messages have the M0 and M1 header registers if the header is present. If the header is not present, all registers below are renumbered starting with M0 where M2 appears. All cases not shown in this table are illegal.

Key:

s0, s1 = source 0, source 1

1/0 = slots 15:8

3/2 = slots 7:0

sZ = source depth

oM = oMask

Message Type	oMask Present	Source Depth Present	Source 0 Alpha Present	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
000	0	0	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A					
000	0	0	1	1/0s 0A	3/2s 0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2 A			
000	0	1	0	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0s Z	3/2s Z			
000	0	1	1	1/0s 0A	3/2s 0A	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2 A	1/0s Z	3/2s Z	
000	1	0	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A				
000	1	0	1	1/0s oA	3/2s 0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0 A	3/2 A		
000	1	1	0	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0A	3/2A	1/0s Z	3/2s Z		
000	1	1	1	1/0s 0A	3/2s 0A	oM	1/0R	3/2R	1/0G	3/2G	1/0B	3/2B	1/0 A	3/2 A	1/0s Z	3/2s Z
001	0	0	0	RGB A												
001	1	0	0	oM	RGB A											
010	0	0	0	1/0s 0R	1/0s 0G	1/0s 0B	1/0s 0A	1/0s 1R	1/0s 1G	1/0s 1B	1/0s 1A					
010	0	1	0	1/0s 0R	1/0s 0G	1/0s 0B	1/0s 0A	1/0s 1R	1/0s 1G	1/0s 1B	1/0s 1A	1/0s Z				

Message Type	oMask Present	Source Depth Present	Source 0 Alpha Present	M2	M3	M4	M5	M6	M7	M8	M9	M10	M11	M12	M13	M14
010	1	0	0	oM	1/0s 0R	1/0s 0G	1/0s 0B	1/0s 0A	1/0s 1R	1/0s 1G	1/0s 1B	1/0s 1A				
010	1	1	0	oM	1/0s 0R	1/0s 0G	1/0s 0B	1/0s 0A	1/0s 1R	1/0s 1G	1/0s 1B	1/0s 1A	1/0s Z			
011	0	0	0	3/2s 0R	3/2s 0G	3/2s 0B	3/2s 0A	3/2s 1R	3/2s 1G	3/2s 1B	3/2s 1A					
011	0	1	0	3/2s 0R	3/2s 0G	3/2s 0B	3/2s 0A	3/2s 1R	3/2s 1G	3/2s 1B	3/2s 1A	3/2s Z				
011	1	0	0	oM	3/2s 0R	3/2s 0G	3/2s 0B	3/2s 0A	3/2s 1R	3/2s 1G	3/2s 1B	3/2s 1A				
011	1	1	0	oM	3/2s 0R	3/2s 0G	3/2s 0B	3/2s 0A	3/2s 1R	3/2s 1G	3/2s 1B	3/2s 1A	3/2s Z			
100	0	0	0	R	G	B	A									
100	0	0	1	s0A	R	G	B	A								
100	0	1	0	R	G	B	A	sZ								
100	0	1	1	s0A	R	G	B	A	sZ							
100	1	0	0	oM	R	G	B	A								
100	1	0	1	s0A	oM	R	G	B	A							
100	1	1	0	oM	R	G	B	A	sZ							
100	1	1	1	s0A	oM	R	G	B	A	sZ						

Single Source

The “normal” render target messages are single source. There are two forms, SIMD16 and SIMD8, intended for the equivalent-sized pixel shader threads. A single color (4 channels) is delivered for each of the 16 or 8 pixels in the message payload. Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

The pixel scoreboard bits corresponding to the dispatched pixel mask (or half of the mask in the case of SIMD8 messages) are cleared only if the **Last Render Target Select** bit is set in the message descriptor.

The single source message does not cause a write to the render target if **Dual Source Blend Enable** in 3DSTATE_WM is *enabled*. However, if **Last Render Target Select** is set, the message still causes pixel scoreboard clear and depth/stencil buffer updates if enabled.

Dual Source

The dual source render target messages only have SIMD8 forms due to maximum message length limitations. SIMD16 pixel shaders must send two of these messages to cover all of the pixels. Each message contains two colors (4 channels each) for each pixel in the message payload. In addition to the first source, the second source can be selected as a blend factor (BLENDFACTOR_*_SRC1_* options in the blend factor fields of COLOR_CALC_STATE or BLEND_STATE). Optional depth, stencil, and antialias alpha information can also be delivered with these messages.

Each dual source message delivered clears the corresponding pixel scoreboard bits if the **Last Render Target Select** bit in the message descriptor is set.

The dual source message reverts to a single source message using source 0 if **Dual Source Blend Enable** in 3DSTATE_WM is disabled.

Replicate Data

The replicate data render target message is used for "fast clear" functionality in cases where the color data for each pixel is identical. This message performs better than the other messages due to its smaller message length. This message does not support depth, stencil, or antialias alpha data being sent with it. This message must target only tiled memory. Access of linear memory using this message type is UNDEFINED. The depth buffer can be cleared through the "early depth" function in conjunction with a pixel shader using this message. Refer to the *Windower* chapter for more details on the early depth function.

The pixel scoreboard bits corresponding to the dispatched pixel mask are cleared only if the **Last Render Target Select** bit is set in the message descriptor.

Multiple Render Targets (MRT)

Multiple render targets are supported with the single source and replicate data messages. Each render target is accessed with a separate Render Target Write message, each with a different surface indicated (different binding table index). The depth buffer is written only by the message(s) to the last render target, indicated by the **Last Render Target Select** bit set to clear the pixel scoreboard bits.

Render Target Read and Write

Render Target Write

This message takes four subspans of pixels for write to a render target. Depending on parameters contained in the message and state, it may also perform a depth and stencil buffer write and/or a render target read for a color blend operation. Additional operations enabled in the Color Calculator state are also initiated as a result of issuing this message (depth test, alpha test, logic ops, etc.). This message is intended only for use by pixel shader kernels for writing results to render targets.

Programming Note	
Context:	Render Target Read and Write

- All surface types, except SURFTYPE_STRBUF, are allowed.
- For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.
- For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a **Render Target Array Index** is included in the input message to provide an additional coordinate. The **Render Target Array Index** must be zero for SURFTYPE_BUFFER.
- The surface format is restricted to the set supported as render target. If source/dest color blend is enabled, the surface format is further restricted to the set supported as alpha blend render target.
- The last message sent to the render target by a thread must have the **End Of Thread** bit set in the message descriptor and the dispatch mask set correctly in the message header to enable correct clearing of the pixel scoreboard.
- The stateless model cannot be used with this message (**Binding Table Index** cannot be 255).
- This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message causes undefined behavior.
- The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG.
- This message cannot be used on a surface in field mode (**Vertical Line Stride** = 1).
- If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same **Slot Group Select** setting.

All surface types, except SURFTYPE_STRBUF, are allowed.

For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.

For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a **Render Target Array Index** is included in the input message to provide an additional coordinate. The **Render Target Array Index** must be zero for SURFTYPE_BUFFER.

The surface format is restricted to the set supported as render target. If source/dest color blend is enabled, the surface format is further restricted to the set supported as alpha blend render target.

The last message sent to the render target by a thread must have the **End Of Thread** bit set in the message descriptor and the dispatch mask set correctly in the message header to enable correct clearing of the pixel scoreboard.

The stateless model cannot be used with this message (**Binding Table Index** cannot be 255).

This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message causes undefined behavior.

The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG.

This message cannot be used on a surface in field mode (**Vertical Line Stride** = 1).

If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same **Slot Group Select** setting.

Project-Specific Restrictions

Programming Note	
Context:	Render Target Read and Write
Execution Mask. For messages without header, the execution mask for render target messages (sent as part of the channel enables on the obus sideband) is used to kill pixels.	

Out-of-Bounds Accesses. Accesses to pixels outside of the surface are dropped and do not modify memory. However, if the **Render Target Array Index** is out of bounds, it is set to zero and the surface write is not surpressed.

The following table indicates the surface formats supported by this message with project restrictions and whether each format supports Alpha Blend.

Surface Format Name	Alpha Blend?
R32G32B32A32_FLOAT	Yes
R32G32B32A32_SINT	No
R32G32B32A32_UINT	No
R16G16B16A16_UNORM	Yes
R16G16B16A16_SNORM	Yes
R16G16B16A16_SINT	No
R16G16B16A16_UINT	No
R16G16B16A16_FLOAT	Yes
R32G32_FLOAT	Yes
R32G32_SINT	No
R32G32_UINT	No
R16G16B16X16_FLOAT	Yes
B8G8R8A8_UNORM	Yes
B8G8R8A8_UNORM_SRGB	Yes
R10G10B10A2_UNORM	Yes
R10G10B10A2_UINT	No
R8G8B8A8_UNORM	Yes
R8G8B8A8_UNORM_SRGB	Yes
R8G8B8A8_SNORM	Yes

Surface Format Name	Alpha Blend?
R8G8B8A8_SINT	No
R8G8B8A8_UINT	No
R16G16_UNORM	Yes
R16G16_SNORM	Yes
R16G16_SINT	No
R16G16_UINT	No
R16G16_FLOAT	Yes
B10G10R10A2_UNORM	Yes
B10G10R10A2_UNORM_SRGB	Yes
R11G11B10_FLOAT	Yes
R32_SINT	No
R32_UINT	No
R32_FLOAT	Yes
B8G8R8X8_UNORM	Yes
B8G8R8X8_UNORM_SRGB	Yes
B5G6R5_UNORM	Yes
B5G6R5_UNORM_SRGB	Yes
B5G5R5A1_UNORM	Yes
B5G5R5A1_UNORM_SRGB	Yes
B4G4R4A4_UNORM	Yes
B4G4R4A4_UNORM_SRGB	Yes
R8G8_UNORM	Yes
R8G8_SNORM	Yes
R8G8_SINT	No
R8G8_UINT	No
R16_UNORM	Yes
R16_SNORM	Yes
R16_SINT	No
R16_UINT	No
R16_FLOAT	Yes
B5G5R5X1_UNORM	Yes
B5G5R5X1_UNORM_SRGB	Yes
A1B5G5R5_UNORM	Yes
A4B4G4R4_UNORM	Yes
R8_UNORM	Yes
R8_SNORM	Yes

Surface Format Name	Alpha Blend?
R8_SINT	No
R8_UINT	No
A8_UNORM	Yes
A1B5G5R5_UNORM	No
A4B4G4R4_UNORM	No
R16G16B16X16_FLOAT	No
R32G32B32X32_FLOAT	No

SubspanPixel to Slot Mapping

The following table indicates the mapping of subspans, pixels, and samples to slots in the pixel shader dispatch depending on the number of samples and message size. This table applies to all devices.

NumSamples Support

NumSamples Support
NumSamples = 1X, 2X, or 4X is supported for all projects.
NumSamples = 8X is supported.

Pixels are numbered as follows within a subspan:

0 = upper left

1 = upper right

2 = lower left

3 = lower right

sspi = Starting Sample Pair Index (from the message header)

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0] Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0]

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0] Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1] Slot[19:16] = Subspan[2].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[2].Pixel[3:0].Sample[1] Slot[27:24] = Subspan[3].Pixel[3:0].Sample[0] Slot[31:28] = Subspan[3].Pixel[3:0].Sample[1]
	4X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0] Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1] Slot[27:24] =

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Subspan[1].Pixel[3:0].Sample[2] Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]
SIMD16	8X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2] Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3] Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4] Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5] Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6] Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]
	1X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0] Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0] Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]
	2X	Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0] Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1] Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0] Slot[15:12] =

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Subspan[1].Pixel[3:0].Sample[1]

Programming Note	
Context:	SubspanPixel to Slot Mapping

- When SIMD32 or SIMD16 PS threads send render target writes with multiple SIMD8 and SIMD16 messages, the following must hold:
- All the slots (as described above) must have a corresponding render target write irrespective of the slot's validity. A slot is considered valid when at least one sample is enabled. For example, a SIMD16 PS thread must send two SIMD8 render target writes to cover all the slots.
- PS thread must send SIMD render target write messages with increasing slot numbers. For example, SIMD16 thread has Slot[15:0] and if two SIMD8 render target writes are used, the first SIMD8 render target write must send Slot[7:0] and the next one must send Slot[15:8].

Message Descriptor

This section contains descriptors for the render target read and write functions.

Message Descriptor - Render Target Write

Message Header

The render target write message has a two-register message header.

Message Header

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td rowspan="2">M0.5</td><td>31:8</td><td>Ignored</td></tr>
<tr><td>7:0</td><td>Dispatch ID. This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread on thread completion.</td></tr>
<tr><td>M0.4</td><td>31:0</td><td>Ignored (reserved for hardware delivery of binding table pointer)</td></tr>
<tr><td>M0.3</td><td>31:0</td><td>Ignored</td></tr>
<tr><td>M0.2</td><td>31:0</td><td>Pixel Mask. One bit per pixel indicating which pixels are lit, possibly impacted by kill instruction activity in the pixel shader. This mask is used to control actual writes to the color buffer. This field is ignored by the read message, all pixels are always returned.
The bits in this mask correspond to the pixels as follows:
<table><tr><td>0</td><td>1</td><td>4</td><td>5</td><td>16</td><td>17</td><td>20</td><td>21</td></tr><tr><td>2</td><td>3</td><td>6</td><td>7</td><td>18</td><td>19</td><td>22</td><td>23</td></tr><tr><td>8</td><td>9</td><td>12</td><td>13</td><td>24</td><td>25</td><td>28</td><td>29</td></tr><tr><td>10</td><td>11</td><td>14</td><td>15</td><td>26</td><td>27</td><td>30</td><td>31</td></tr></table></td></tr>
<tr><td>M0.1</td><td>31:0</td><td>Y offset. The Y offset of the upper left corner of the block into the surface. Must be 4-row aligned (Bits 1:0 MBZ).
Format = S31</td></tr>
<tr><td>M0.0</td><td>31:0</td><td>X offset. The X offset of the upper left corner of the block into the surface. This is a pixel offset assuming a 32-bit pixel. Must be 8-pixel aligned (Bits 2:0 MBZ).
Format = S31</td></tr>
</table>

Writeback Message (Read)

A SIMD16 writeback message consists of up to 8 destination registers. If a channel/component is not present in the RT format, the corresponding GRF is filled with zeroes or 1.0 in float/int depending on whether RGB or Alpha are disabled.

DWord	Bits	Description
W0.7	31:0	Slot 7 Red. Specifies the value of the red channel for slot 7. Format = 32 bits raw data.
W0.6	31:0	Slot 6 Red
W0.5	31:0	Slot 5 Red
W0.4	31:0	Slot 4 Red
W0.3	31:0	Slot 3 Red
W0.2	31:0	Slot 2 Red
W0.1	31:0	Slot 1 Red
W0.0	31:0	Slot 0 Red
W1.7	31:0	Slot 15 Red
W1.6	31:0	Slot 14 Red
W1.5	31:0	Slot 13 Red
W1.4	31:0	Slot 12 Red
W1.3	31:0	Slot 11 Red
W1.2	31:0	Slot 10 Red
W1.1	31:0	Slot 9 Red
W1.0	31:0	Slot 8 Red
W2		Slots 7:0 Green
W3		Slots 15:8 Green
W4		Slots 7:0 Blue
W5		Slots 15:8 Blue
W6		Slots 7:0 Alpha
W7		Slots 15:8 Alpha

A SIMD8 writeback message consists of up to 4 destination registers. Which registers are returned is determined by the channel mask in the message descriptor. Each asserted channel mask results in the destination register of the corresponding channel being filled with zeroes or 1.0 in float/int depending on whether RGB or Alpha are disabled.

DWord	Bits	Description
W0.7	31:0	Slot 7 Red. Specifies the value of the red channel for slot 7. Format = 32 bits raw data.
W0.6	31:0	Slot 6 Red
W0.5	31:0	Slot 5 Red
W0.4	31:0	Slot 4 Red
W0.3	31:0	Slot 3 Red
W0.2	31:0	Slot 2 Red
W0.1	31:0	Slot 1 Red
W0.0	31:0	Slot 0 Red
W1		Slots 7:0 Green
W2		Slots 7:0 Blue
W3		Slots 7:0 Alpha

Header for SIMD8_IMAGE_WRITE

DWord	Bits	Description
M0.5	31:10	Ignored
	9:8	**Color Code:** This ID is assigned by the Windower unit and is used to track synchronizng events. Format: Reserved for HW Implementation Use.
	7:0	**FFTID.** The Fixed Function Thread ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31:0	Ignored
M0.2	31:3	Ignored
	2:0	**Render Target Index.** Specifies the render target index that will be used to select blend state from BLEND_STATE. Format = U3
M0.1	31:6	**ColorCalculatorState Pointer.** Specifies the 64-byte aligned pointer to the color calculator state. This pointer is relative to the **General State Base Address**. Format = GeneralStateOffset[31:6]

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>For SIMD8_IMAGE_WR message under normal GPGPU usage model, we recommend that SW program a dummy color-calc state such that all operations controlled by this state are disabled.</td></tr>
<tr><td></td><td>5:0</td><td>Ignored</td></tr>
<tr><td>M0.0</td><td>31</td><td>Ignored</td></tr>
<tr><td></td><td>30:27</td><td>Viewport Index. Specifies the index of the viewport currently being used.
Format = U4
Range = [0,15]
SIMD8_IMAGE_WR message type this field is ignored by hardware.</td></tr>
<tr><td></td><td>26:16</td><td>Render Target Array Index. Specifies the array index to be used for the following surface types:
SURFTYPE_1D: specifies the array index. Range = [0,511]
SURFTYPE_2D: specifies the array index. Range = [0,511]
SURFTYPE_3D: specifies the “z” or “r” coordinate. Range = [0,2047]
SURFTYPE_CUBE: specifies the face identifier. Range = [0,5]
SURFTYPE_BUFFER: must be zero.
<table>
<tr><th>face</th><th>Render Target Array Index</th></tr>
<tr><td>+x</td><td>0</td></tr>
<tr><td>-x</td><td>1</td></tr>
<tr><td>+y</td><td>2</td></tr>
<tr><td>-y</td><td>3</td></tr>
<tr><td>+z</td><td>4</td></tr>
<tr><td>-z</td><td>5</td></tr>
</table>
Format = U11
The Render Target Array Index used by hardware for access to the Render Target is overridden with the Minimum Array Element defined in SURFACE_STATE if it is out of the range between Minimum Array Element and Depth. For cube surfaces, a depth value of 5 is used for this determination.
For SMD8_IMAGE_WRITE:
For SURFTYPE_2D, this field must be 0.
For SURFTYPE_3D, this field may not be 0 for "Write-3D-Image" operation.</td></tr>
<tr><td></td><td>15:8</td><td>Ignored</td></tr>
<tr><td></td><td>7:0</td><td>Pixel Masks for SIMD8 messages.
1: Pixel is enabled.</td></tr>
</table>

DWord	Bits	Description
		0: Pixel is disabled. In this case the corresponding (x,y) should be ignored by hardware.
M1.7	31:16	Y7: y-coordinate for pixel 7 Format = U16
	15:0	X7: x-coordinate for pixel 7 Format = U16
M1.6	31:16	Y6: y-coordinate for pixel 6 Format = U16
	15:0	X6: x-coordinate for pixel 6 Format = U16
M1.5	31:16	Y5: y-coordinate for pixel 5 Format = U16
	15:0	X5: x-coordinate for pixel 5 Format = U16
M1.4	31:16	Y4: y-coordinate for pixel 4 Format = U16
	15:0	X4: x-coordinate for pixel 4 Format = U16
M1.3	31:16	Y3: y-coordinate for pixel 3 Format = U16
	15:0	X3: x-coordinate for pixel 3 Format = U16
M1.2	31:16	Y2: y-coordinate for pixel 2 Format = U16
	15:0	X2: x-coordinate for pixel 2 Format = U16

DWord	Bits	Description
M1.1	31:16	Y1: y-coordinate for pixel 1 Format = U16
	15:0	X1: x-coordinate for pixel 1 Format = U16
M1.0	31:16	Y0: y-coordinate for pixel 0 Format = U16
	15:0	X0: x-coordinate for pixel 0 Format = U16

Source 0 Alpha Payload

The source 0 alpha registers, if included, appear in M2 and M3, immediately following the header (if present).

For the SIMD8 single source message, only slot 7:0 data is sent (M2). The source 0 alpha phases are not supported for dual source messages.

DWord	Bit	Description
M2.7	31:0	Source 0 Alpha for Slot 7 Format = IEEE_Float This and the next register is only included if **Source 0 Alpha Present** bit is set.
M2.6	31:0	Source 0 Alpha for Slot 6
M2.5	31:0	Source 0 Alpha for Slot 5
M2.4	31:0	Source 0 Alpha for Slot 4
M2.3	31:0	Source 0 Alpha for Slot 3
M2.2	31:0	Source 0 Alpha for Slot 2
M2.1	31:0	Source 0 Alpha for Slot 1
M2.0	31:0	Source 0 Alpha for Slot 0
M3.7	31:0	Source 0 Alpha for Slot 15
M3.6	31:0	Source 0 Alpha for Slot 14
M3.5	31:0	Source 0 Alpha for Slot 13
M3.4	31:0	Source 0 Alpha for Slot 12
M3.3	31:0	Source 0 Alpha for Slot 11
M3.2	31:0	Source 0 Alpha for Slot 10
M3.1	31:0	Source 0 Alpha for Slot 9
M3.0	31:0	Source 0 Alpha for Slot 8

oMask Payload

The oMask payload, if present, follows source 0 alpha. The value of 'p' depends on whether the header and source 0 alpha are present.

Sample "n" for that pixel will be killed (not written to the render target or depth buffer) if bit "n" of the oMask is zero. Bits numbers where "n" is larger than the number of multisamples are ignored.

For the SIMD8 messages, only slots 7:0 data is used, or only slots 15:8 depending on the **Message Type** encoding.

DWord	Bit	Description
Mp.7	31:16	oMask for Slot 15 Format = 16-bit mask This register is only included if **oMask Present** bit is set.
	15:0	oMask for Slot 14
Mp.6	31:16	oMask for Slot 13
	15:0	oMask for Slot 12
Mp.5	31:16	oMask for Slot 11
	15:0	oMask for Slot 10
Mp.4	31:16	oMask for Slot 9
	15:0	oMask for Slot 8
Mp.3	31:16	oMask for Slot 7
	15:0	oMask for Slot 6
Mp.2	31:16	oMask for Slot 5
	15:0	oMask for Slot 4
Mp.1	31:16	oMask for Slot 3
	15:0	oMask for Slot 2
Mp.0	31:16	oMask for Slot 1
	15:0	oMask for Slot 0

Color Payload SIMD16 Single Source

Color Payload

This payload is included if the Message Type is SIMD16 single source. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

DWord	Bits	Description
Mm.7	31:0	**Slot 7 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	Slot 6 Red
Mm.5	31:0	Slot 5 Red
Mm.4	31:0	Slot 4 Red
Mm.3	31:0	Slot 3 Red
Mm.2	31:0	Slot 2 Red
Mm.1	31:0	Slot 1 Red
Mm.0	31:0	Slot 0 Red
M(m+1).7	31:0	Slot 15 Red
M(m+1).6	31:0	Slot 14 Red
M(m+1).5	31:0	Slot 13 Red
M(m+1).4	31:0	Slot 12 Red
M(m+1).3	31:0	Slot 11 Red
M(m+1).2	31:0	Slot 10 Red
M(m+1).1	31:0	Slot 9 Red
M(m+1).0	31:0	Slot 8 Red
M(m+2)		**Slot[7:0] Green.** See Mm definition for slot locations.
M(m+3)		**Slot[15:8] Green.** See M(m+1) definition for slot locations.
M(m+4)		**Slot[7:0] Blue.** See Mm definition for slot locations.
M(m+5)		**Slot[15:8] Blue.** See M(m+1) definition for slot locations.
M(m+6)		**Slot[7:0] Alpha.** See Mm definition for slot locations.
M(m+7)		**Slot[15:8] Alpha.** See M(m+1) definition for slot locations.

Color Payload: CHV, BSW

DWord	Bits	Description
Mm.7	31:0	Slot 15 Red Slot 14 Red
Mm.6	31:0	Slot 13 Red Slot 12 Red
Mm.5	31:0	Slot 11 Red Slot 10 Red
Mm.4	31:0	Slot 9 Red Slot 8 Red
Mm.3	31:0	Slot 7 Red Slot 6 Red
Mm.2	31:0	Slot 5 Red Slot 4 Red
Mm.1	31:0	Slot 3 Red Slot 2 Red
Mm.0	31:16 15:0	Slot 1 Red Slot 0 Red
M(m+1)		**Slot[15:0] Green.** See Mm definition for slot locations.
M(m+2)		**Slot[15:0] Blue.** See Mm definition for slot locations.
M(m+3)		**Slot[15:0] Alpha.** See M(m+1) definition for slot locations.

Color Payload SIMD8 Single Source

This payload is included if the Message Type is SIMD8 single source or SIMD8 Image Write. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

DWord	Bits	Description
Mm.7	31:0	**Slot 7 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	Slot 6 Red
Mm.5	31:0	Slot 5 Red
Mm.4	31:0	Slot 4 Red
Mm.3	31:0	Slot 3 Red
Mm.2	31:0	Slot 2 Red
Mm.1	31:0	Slot 1 Red
Mm.0	31:0	Slot 0 Red
M(m+1)		**Slot[7:0] Green.** See Mm definition for slot locations.
M(m+2)		**Slot[7:0] Blue.** See Mm definition for slot locations.
M(m+3)		**Slot[7:0] Alpha.** See Mm definition for slot locations.

Color Payload: SIMD8 Single Source: CHV, BSW

This payload is included if the Message Type is SIMD8 single source or SIMD8 Image Write. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

DWord	Bits	Description
Mm.3	31:16	Slot 7 Red
	15:0	Slot 6 Red
Mm.2	31:16	Slot 5 Red
	15:0	Slot 4 Red
Mm.1	31:16	Slot 3 Red
	15:0	Slot 2 Red
Mm.0	31:16	Slot 1 Red
	15:0	Slot 0 Red
M(m+1)		**Slot[7:0] Green.** See Mm definition for slot locations.
M(m+2)		**Slot[7:0] Blue.** See Mm definition for slot locations.
M(m+3)		**Slot[7:0] Alpha.** See M(m+1) definition for slot locations.

Color Payload SIMD16 Replicated Data

This payload is included if the Message Type specifies a single source message with replicated data. One set of R/G/B/A data is included in the message, and this data is replicated to all 16 pixels.

This message is legal with color data; oMask is also legal with this message. The registers for depth, stencil, and antialias alpha data cannot be included with this message, and the corresponding bits in the message header must indicate that these registers are not present.

The value of 'm' depends on whether the header and oMask are present.

Note: This message is allowed only on tiled surfaces.

DWord	Bits	Description
Mm.7:4	31:0	Reserved
Mm.3	31:0	**Alpha.** Specifies the value of the alpha channel for all slots. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.2	31:0	Blue
Mm.1	31:0	Green
Mm.0	31:0	Red

Color Payload: SIMD16 Replicated Data: CHV, BSW

This payload is included if the Message Type specifies single source message with replicated data. One set of R/G/B/A data is included in the message, and this data is replicated to all 16 pixels.

This message is legal with color data only; oMask is also legal with this message. The registers for depth, stencil, and antialias alpha data cannot be included with this message, and the corresponding bits in the message header must indicate that these registers are not present.

The value of 'm' depends on whether the header and oMask are present.

Note: This message is allowed only on tiled surfaces.

DWord	Bits	Description
Mm.7:2	31:0	Reserved
Mm.1	31:16	Alpha
	15:0	Blue
Mm.0	31:16	Green
	15:0	Red

Color Payload SIMD8 Dual Source

This payload is included if the **Message Type** specifies dual source message. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

Programming Note	
Context:	Color Payload SIMD8 Dual Source
The dual source message contains only 2 subspans (8 pixels) due to limitations in message length.	

DWord	Bits	Description
Mm.7	31:0	**Slot 7 Source 0 Red.** Specifies the value of the slot's red component. Format = IEEE Float, S31, or U32 depending on the **Surface Format** of the surface being accessed. SINT formats use S31, UINT formats use U32, and all other formats use Float.
Mm.6	31:0	Slot 6 Source 0 Red
Mm.5	31:0	Slot 5 Source 0 Red
Mm.4	31:0	Slot 4 Source 0 Red
Mm.3	31:0	Slot 3 Source 0 Red
Mm.2	31:0	Slot 2 Source 0 Red
Mm.1	31:0	Slot 1 Source 0 Red
Mm.0	31:0	Slot 0 Source 0 Red
M(m+1)		**Slot[7:0] Source 0 Green.** See Mm definition for slot locations.
M(m+2)		**Slot[7:0] Source 0 Blue.** See Mm definition for slot locations.
M(m+3)		**Slot[7:0] Source 0 Alpha.** See Mm definition for slot locations.
M(m+4)		**Slot[7:0] Source 1 Red.** See Mm definition for slot locations.
M(m+5)		**Slot[7:0] Source 1 Green.** See Mm definition for slot locations.
M(m+6)		**Slot[7:0] Source 1 Blue.** See Mm definition for slot locations.
M(m+7)		**Slot[7:0] Source 1 Alpha.** See Mm definition for slot locations.

Color Payload: SIMD8 Dual Source: CHV, BSW

This payload is included if the **Message Type** specifies dual source message. The value of 'm' depends on whether the header, source 0 alpha, and oMask are present.

The dual source message contains only 2 subspans (8 pixels) due to limitations in message length.

DWord	Bits	Description
Mm.7	31:16	Slot 7 Source 1 Red
	15:0	Slot 6 Source 1 Red
Mm.6	31:16	Slot 5 Source 1 Red
	15:0	Slot 4 Source 1 Red
Mm.5	31:16	Slot 3 Source 1 Red
	15:0	Slot 2 Source 1 Red
Mm.4	31:16	Slot 1 Source 1 Red
	15:0	Slot 0 Source 1 Red
Mm.3	31:16	Slot 7 Source 0 Red
	15:0	Slot 6 Source 0 Red
Mm.2	31:16	Slot 5 Source 0 Red
	15:0	Slot 4 Source 0 Red
Mm.1	31:16	Slot 3 Source 0 Red
	15:0	Slot 2 Source 0 Red
Mm.0	31:16	Slot 1 Source 0 Red
	15:0	Slot 0 Source 0 Red
M(m+1)		**Slot[7:0] Green.** See Mm definition for slot locations.
M(m+2)		**Slot[7:0] Blue.** See Mm definition for slot locations.
M(m+3)		**Slot[7:0] Alpha.** See Mm definition for slot locations.

Render Target Read and Write

Render Target Write

This message takes four subspans of pixels for write to a render target. Depending on parameters contained in the message and state, it may also perform a depth and stencil buffer write and/or a render target read for a color blend operation. Additional operations enabled in the Color Calculator state are also initiated as a result of issuing this message (depth test, alpha test, logic ops, etc.). This message is intended only for use by pixel shader kernels for writing results to render targets.

Programming Note	
Context:	Render Target Read and Write

- All surface types, except SURFTYPE_STRBUF, are allowed.
- For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.
- For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a **Render Target Array Index** is included in the input message to provide an additional coordinate. The **Render Target Array Index** must be zero for SURFTYPE_BUFFER.
- The surface format is restricted to the set supported as render target. If source/dest color blend is enabled, the surface format is further restricted to the set supported as alpha blend render target.
- The last message sent to the render target by a thread must have the **End Of Thread** bit set in the message descriptor and the dispatch mask set correctly in the message header to enable correct clearing of the pixel scoreboard.
- The stateless model cannot be used with this message (**Binding Table Index** cannot be 255).
- This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message causes undefined behavior.
- The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG.
- This message cannot be used on a surface in field mode (**Vertical Line Stride** = 1).
- If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same **Slot Group Select** setting.

All surface types, except SURFTYPE_STRBUF, are allowed.

For SURFTYPE_BUFFER and SURFTYPE_1D surfaces, only the X coordinate is used to index into the surface. The Y coordinate must be zero.

For SURFTYPE_1D, 2D, 3D, and CUBE surfaces, a **Render Target Array Index** is included in the input message to provide an additional coordinate. The **Render Target Array Index** must be zero for SURFTYPE_BUFFER.

The surface format is restricted to the set supported as render target. If source/dest color blend is enabled, the surface format is further restricted to the set supported as alpha blend render target.

The last message sent to the render target by a thread must have the **End Of Thread** bit set in the message descriptor and the dispatch mask set correctly in the message header to enable correct clearing of the pixel scoreboard.

The stateless model cannot be used with this message (**Binding Table Index** cannot be 255).

This message can only be issued from a kernel specified in WM_STATE or 3DSTATE_WM (pixel shader kernel), dispatched in non-contiguous mode. Any other kernel issuing this message causes undefined behavior.

The dual source message cannot be used if the Render Target Rotation field in SURFACE_STATE is set to anything other than RTROTATE_0DEG.

This message cannot be used on a surface in field mode (**Vertical Line Stride** = 1).

If multiple SIMD8 Dual Source messages are delivered by the pixel shader thread, each SIMD8_DUALSRC_LO message must be issued *before* the SIMD8_DUALSRC_HI message with the same **Slot Group Select** setting.

Project-Specific Restrictions

Programming Note	
Context:	Render Target Read and Write
Execution Mask. For messages without header, the execution mask for render target messages (sent as part of the channel enables on the obus sideband) is used to kill pixels.	

Out-of-Bounds Accesses. Accesses to pixels outside of the surface are dropped and do not modify memory. However, if the **Render Target Array Index** is out of bounds, it is set to zero and the surface write is not surpressed.

The following table indicates the surface formats supported by this message with project restrictions and whether each format supports Alpha Blend.

Surface Format Name	Alpha Blend?
R32G32B32A32_FLOAT	Yes
R32G32B32A32_SINT	No
R32G32B32A32_UINT	No
R16G16B16A16_UNORM	Yes
R16G16B16A16_SNORM	Yes
R16G16B16A16_SINT	No
R16G16B16A16_UINT	No
R16G16B16A16_FLOAT	Yes
R32G32_FLOAT	Yes
R32G32_SINT	No
R32G32_UINT	No
R16G16B16X16_FLOAT	Yes
B8G8R8A8_UNORM	Yes

Surface Format Name	Alpha Blend?
B8G8R8A8_UNORM_SRGB	Yes
R10G10B10A2_UNORM	Yes
R10G10B10A2_UINT	No
R8G8B8A8_UNORM	Yes
R8G8B8A8_UNORM_SRGB	Yes
R8G8B8A8_SNORM	Yes
R8G8B8A8_SINT	No
R8G8B8A8_UINT	No
R16G16_UNORM	Yes
R16G16_SNORM	Yes
R16G16_SINT	No
R16G16_UINT	No
R16G16_FLOAT	Yes
B10G10R10A2_UNORM	Yes
B10G10R10A2_UNORM_SRGB	Yes
R11G11B10_FLOAT	Yes
R32_SINT	No
R32_UINT	No
R32_FLOAT	Yes
B8G8R8X8_UNORM	Yes
B8G8R8X8_UNORM_SRGB	Yes
B5G6R5_UNORM	Yes
B5G6R5_UNORM_SRGB	Yes
B5G5R5A1_UNORM	Yes
B5G5R5A1_UNORM_SRGB	Yes
B4G4R4A4_UNORM	Yes
B4G4R4A4_UNORM_SRGB	Yes
R8G8_UNORM	Yes
R8G8_SNORM	Yes
R8G8_SINT	No
R8G8_UINT	No
R16_UNORM	Yes
R16_SNORM	Yes
R16_SINT	No
R16_UINT	No
R16_FLOAT	Yes

Surface Format Name	Alpha Blend?
B5G5R5X1_UNORM	Yes
B5G5R5X1_UNORM_SRGB	Yes
A1B5G5R5_UNORM	Yes
A4B4G4R4_UNORM	Yes
R8_UNORM	Yes
R8_SNORM	Yes
R8_SINT	No
R8_UINT	No
A8_UNORM	Yes
A1B5G5R5_UNORM	No
A4B4G4R4_UNORM	No
R16G16B16X16_FLOAT	No
R32G32B32X32_FLOAT	No

Message Header

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td rowspan="2">M0.5</td><td>31:8</td><td>Ignored</td></tr>
<tr><td>7:0</td><td>Dispatch ID. This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread on thread completion.</td></tr>
<tr><td>M0.4</td><td>31:0</td><td>Ignored (reserved for hardware delivery of binding table pointer)</td></tr>
<tr><td>M0.3</td><td>31:0</td><td>Ignored</td></tr>
<tr><td>M0.2</td><td>31:0</td><td>Pixel Mask. One bit per pixel indicating which pixels are lit, possibly impacted by kill instruction activity in the pixel shader. This mask is used to control actual writes to the color buffer. This field is ignored by the read message, all pixels are always returned.
The bits in this mask correspond to the pixels as follows:
<table><tr><td>0</td><td>1</td><td>4</td><td>5</td><td>16</td><td>17</td><td>20</td><td>21</td></tr><tr><td>2</td><td>3</td><td>6</td><td>7</td><td>18</td><td>19</td><td>22</td><td>23</td></tr><tr><td>8</td><td>9</td><td>12</td><td>13</td><td>24</td><td>25</td><td>28</td><td>29</td></tr><tr><td>10</td><td>11</td><td>14</td><td>15</td><td>26</td><td>27</td><td>30</td><td>31</td></tr></table></td></tr>
<tr><td>M0.1</td><td>31:0</td><td>Y offset. The Y offset of the upper left corner of the block into the surface. Must be 4-row aligned (Bits 1:0 MBZ).
Format = S31</td></tr>
<tr><td>M0.0</td><td>31:0</td><td>X offset. The X offset of the upper left corner of the block into the surface. This is a pixel offset assuming a 32-bit pixel. Must be 8-pixel aligned (Bits 2:0 MBZ).
Format = S31</td></tr>
</table>

Shared Functions Pixel Interpolater

The Pixel Interpolator provides barycentric parameters at various offsets relative to the pixel location. These barycentric parameters are in the same format and layout as those received in the pixel shader dispatch. Please refer to the “Windower” chapter in the “3D Pipeline” volume for more details on barycentric parameters.

Barycentric parameters delivered in the pixel shader payload are at pre-defined positions based on **Barycentric Interpolation Mode** bits selected in 3DSTATE_WM. The pixel interpolator allows barycentric parameters to be computed at additional locations.

Messages

The following is the message definition for the Pixel Interpolator shared function.

Programming Note	
Context:	Messages
Pixel Interpolator messages can only be delivered by pixel shader kernels.	

Execution Mask. Each bit in the execution mask enables the corresponding slot’s barycentric parameter return to the destination registers.

Initiating Message

This topic is currently under development.

Message Descriptor

Bits	Description
19	**Header Present:** Specifies whether the message includes a header phase. Must be zero for all *Pixel Interpolator* messages. Format = Enable
18:17	Ignored
16	**SIMD Mode.** Specifies the SIMD mode of the message being sent. Format = U1 0: SIMD8 mode 1: SIMD16 mode
15	Ignored
14	**Interpolation Mode.** Specifies which interpolation mode is used. Format = U1 0: Perspective Interpolation

Bits	Description
	1: Linear Interpolation **Programming Note** **Context:** Message Descriptor This field cannot be set to "Linear Interpolation" unless **Non-Perspective Barycentric Enable** in 3DSTATE_CLIP is enabled.
13:12	**Message Type.** Specifies the type of message being sent when pixel-rate evaluation requested. Format = U2 0: Per Message Offset (eval_snapped with immediate offset) 1: Sample Position Offset (eval_sindex) 2: Centroid Position Offset (eval_centroid) 3: Per Slot Offset (eval_snapped with register offset) **Programming Note** **Context:** Message Descriptor When Message Type is Sample Position, requesting an attribute at sample index beyond the range defined by the Forced Sample Count (aka NUM_RASTSAMPLES) is illegal.
11	**Slot Group Select.** This field selects whether slots 15:0 or slots 31:16 are used for bypassed data. Bypassed data includes the X/Y addresses and centroid position. For 8- and 16-pixel dispatches, SLOTGRP_LO must be selected on every message. For 32-pixel dispatches, this field must be set correctly for each message based on which slots are currently being processed. 0: SLOTGRP_LO: Choose bypassed data for slots 15:0. 1: SLOTGRP_HI: Choose bypassed data for slots 31:16. **Programming Note** **Context:** Message Descriptor This field must be set to SLOTGRP_LO for SIMD8 messages. SIMD8 messages always use bypassed data for slots 7:0.
10:8	Ignored
7:0	**Message Specific Control.** Refer to the sections below for the definition of these bits based on **Message Type**.

"Per Message Offset" Message Descriptor

Bit	Description
7:4	**Per Message Y Pixel Offset** Specifies the Y Pixel Offset that applies to all slots. Format = S4 2's complement representing units of 1/16 pixel. Range = [-8/16, +7/16]
3:0	**Per Message X Pixel Offset** Specifies the X Pixel Offset that applies to all slots. Format = S4 2's complement representing units of 1/16 pixel. Range = [-8/16, +7/16]

"Sample Position Offset" Message Descriptor

Bits	Description
7:4	**Sample Index** Specifies the sample index that applies to all slots. [CHV, BSW]: Sample Index must not exceed the value of NUM_RASTSAMPLES when NUM_RASTSAMPLES > 1. From API, perspective, Forced Sample Count Defines the maximum allowable index in this message. Format = U4 Range: [0, 15]
3:0	Ignored

"Centroid Position" and "Per Slot Offset" Message Descriptor

Bit	Description
7:0	Ignored

Message Payload for Most Messages

This message payload applies to the following message types:

- Per Message Offset
- Sample Position Offset
- Centroid Position Offset

DWord	Bit	Description
M0.7:0		Ignored

SIMD8 Per Slot Offset Message Payload

This message payload applies only to the SIMD8 Per Slot Offset message type. The message length is 2.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**
M1.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M1.6	31:0	**Slot 6 Y Pixel Offset**
M1.5	31:0	**Slot 5 Y Pixel Offset**
M1.4	31:0	**Slot 4 Y Pixel Offset**
M1.3	31:0	**Slot 3 Y Pixel Offset**
M1.2	31:0	**Slot 2 Y Pixel Offset**
M1.1	31:0	**Slot 1 Y Pixel Offset**
M1.0	31:0	**Slot 0 Y Pixel Offset**

SIMD16 Per Slot Offset Message Payload

This message payload applies only to the SIMD16 Per Slot Offset message type. The message length is 4.

DWord	Bit	Description
M0.7	31:0	**Slot 7 X Pixel Offset** Specifies the X pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M0.6	31:0	**Slot 6 X Pixel Offset**
M0.5	31:0	**Slot 5 X Pixel Offset**
M0.4	31:0	**Slot 4 X Pixel Offset**
M0.3	31:0	**Slot 3 X Pixel Offset**
M0.2	31:0	**Slot 2 X Pixel Offset**
M0.1	31:0	**Slot 1 X Pixel Offset**
M0.0	31:0	**Slot 0 X Pixel Offset**
M1.7	31:0	**Slot 15 X Pixel Offset**
M1.6	31:0	**Slot 14 X Pixel Offset**
M1.5	31:0	**Slot 13 X Pixel Offset**
M1.4	31:0	**Slot 12 X Pixel Offset**
M1.3	31:0	**Slot 11 X Pixel Offset**
M1.2	31:0	**Slot 10 X Pixel Offset**
M1.1	31:0	**Slot 9 X Pixel Offset**
M1.0	31:0	**Slot 8 X Pixel Offset**
M2.7	31:0	**Slot 7 Y Pixel Offset** Specifies the Y pixel offset for slot 7. Format = S4 2's complement representing units of 1/16 pixel. The upper 28 bits are ignored. Range = [-8/16, +7/16]
M2.6	31:0	**Slot 6 Y Pixel Offset**
M2.5	31:0	**Slot 5 Y Pixel Offset**
M2.4	31:0	**Slot 4 Y Pixel Offset**
M2.3	31:0	**Slot 3 Y Pixel Offset**
M2.2	31:0	**Slot 2 Y Pixel Offset**
M2.1	31:0	**Slot 1 Y Pixel Offset**
M2.0	31:0	**Slot 0 Y Pixel Offset**
M3.7	31:0	**Slot 15 Y Pixel Offset**

DWord	Bit	Description
M3.6	31:0	**Slot 14 Y Pixel Offset**
M3.5	31:0	**Slot 13 Y Pixel Offset**
M3.4	31:0	**Slot 12 Y Pixel Offset**
M3.3	31:0	**Slot 11 Y Pixel Offset**
M3.2	31:0	**Slot 10 Y Pixel Offset**
M3.1	31:0	**Slot 9 Y Pixel Offset**
M3.0	31:0	**Slot 8 Y Pixel Offset**

Writeback Message

This topic is currently under development.

SIMD8

The response length for all SIMD8 messages is 2. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**

DWord	Bit	Description
W1.4	31:0	**Barycentric[2] for Slot 4**
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0**

SIMD16

The response length for all SIMD16 messages is 4. The data for each slot is written only if its corresponding execution mask bit is set.

DWord	Bit	Description
W0.7	31:0	**Barycentric[1] for Slot 7** Format = IEEE_Float
W0.6	31:0	**Barycentric[1] for Slot 6**
W0.5	31:0	**Barycentric[1] for Slot 5**
W0.4	31:0	**Barycentric[1] for Slot 4**
W0.3	31:0	**Barycentric[1] for Slot 3**
W0.2	31:0	**Barycentric[1] for Slot 2**
W0.1	31:0	**Barycentric[1] for Slot 1**
W0.0	31:0	**Barycentric[1] for Slot 0**
W1.7	31:0	**Barycentric[2] for Slot 7** Format = IEEE_Float
W1.6	31:0	**Barycentric[2] for Slot 6**
W1.5	31:0	**Barycentric[2] for Slot 5**
W1.4	31:0	**Barycentric[2] for Slot 4**

DWord	Bit	Description
W1.3	31:0	**Barycentric[2] for Slot 3**
W1.2	31:0	**Barycentric[2] for Slot 2**
W1.1	31:0	**Barycentric[2] for Slot 1**
W1.0	31:0	**Barycentric[2] for Slot 0** Format = IEEE_Float
W2.7	31:0	**Barycentric[1] for Slot 15**
W2.6	31:0	**Barycentric[1] for Slot 14**
W2.5	31:0	**Barycentric[1] for Slot 13**
W2.4	31:0	**Barycentric[1] for Slot 12**
W2.3	31:0	**Barycentric[1] for Slot 11**
W2.2	31:0	**Barycentric[1] for Slot 10**
W2.1	31:0	**Barycentric[1] for Slot 9**
W2.0	31:0	**Barycentric[1] for Slot 8**
W3.7	31:0	**Barycentric[2] for Slot 15**
W3.6	31:0	**Barycentric[2] for Slot 14**
W3.5	31:0	**Barycentric[2] for Slot 13**
W3.4	31:0	**Barycentric[2] for Slot 12**
W3.3	31:0	**Barycentric[2] for Slot 11**
W3.2	31:0	**Barycentric[2] for Slot 10**
W3.1	31:0	**Barycentric[2] for Slot 9**
W3.0	31:0	**Barycentric[2] for Slot 8**

Shared Functions - Unified Return Buffer (URB)

The Unified Return Buffer (URB) is a general-purpose buffer used for sending data between different threads, and, in some cases, between threads and fixed-function units (or vice-versa). A thread accesses the URB by sending messages.

URB Size

An URB entry is a logical entity within the URB, referenced by an entry handle and comprised of some number of consecutive rows. A row corresponds in size to a 256-bit EU GRF register. Read/write access to the URB is generally supported on a row-granular basis.

URB Size	URB Rows	URB Rows when SLM Enabled
See the Configurations volume.		

URB Access

The URB can be written by the following agents:

- Command Stream (CS) can write constant data into Constant URB Entries (CURBEs) as a result of processing CONSTANT_BUFFER commands.
- The Video Front End (VFE) fixed-function unit of the Media pipeline can write thread payload data in to its URB entries.
- The Vertex Fetch (VF) fixed-function unit of the 3D pipeline can write vertex data into its URB entries.
- GEN4 threads can write data into URB entries via URB_WRITE messages sent to the URB shared function.

The URB can be read by the following agents:

- The Thread Dispatcher (TD) is the main source of URB reads. As a part of spawning a thread, pipeline fixed-functions provide the TD with a number of URB handles, read offsets, and lengths. The TD reads the specified data from the URB and provide that data in the thread payload pre-loaded into GRF registers.
- The Geometry Shader (GS) and Clipper (CLIP) fixed-function units of the 3D pipeline can read selected parts of URB entries to extract vertex data required by the pipeline.
- The Windower (WM) FF unit reads back depth coefficients from URB entries written by the Strip/Fan unit.

Programming Note	
Context:	URB Access
The CPU cannot read the URB directly.	

State

The URB function is stateless, with all information required to perform a function being passed in the write message.

See URB Allocation (*Graphics Processing Engine*) for a discussion of how the URB is divided amongst the various fixed functions.

FF_SYNC Messages

FF_SYNC messages pass critical information between GS/Clip threads and the GS/Clip FF units, as well as providing GS/Clip thread synchronization (ordering). GS threads report various counts resulting from running the GS and/or SO functions, prior to performing any output (to SOB buffers or to URB handles). Clip threads report only the number of handles required. A message response (writeback) length of 1 GRF is indicated on the 'send' instruction if the thread requires response data and/or synchronization. Refer to the GS/Clip stage chapter for details.

FF_SYNC Message Header

DWord	Bits	Description
M0.5	31:8	Ignored
	7:0	**FFTID.** The Fixed Function Thread ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread on thread completion.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:7	Ignored
M0.1	31:16	Ignored
	15:0	(GS-only) **NumGSPrimsGenerated.** The number of objects (e.g., triangles) generated by the GS function performed by the thread. If the GS function is not enabled, this field MBZ. Format: U16 Range: [0,1024]
M0.0	31:16	(GS-only) **NumSOVertsToWrite**. The number of (expanded-to-list) vertices generated by the SO function performed by the thread. This represents the number of vertices the thread attempts to write to the SOBs in memory, once it obtains the SVBIs in the FF_SYNC writeback. Note that overflow may occur either (a) prior to the SVBIs being returned in the writeback or (b) in the process of this thread outputting to the SOBs. In either case, the thread needs to check for overflow once it receives the writeback, based on the returned SVBIs and the number of vertices it must attempt to output. If the SO function is not enabled, this field MBZ. This field is ignored if the **SVBI Post Increment** field is set in the GS_3DSTATE, described in the 3D pipeline volumes.

DWord	Bits	Description
		Format: U16 Range: [0,3066] (1024-vertex tristrip = 1022 triangles = 3066 trilist vertices)
	15:0	(GS-only) **NumSOPrimsNeeded.** The number of objects (e.g., triangles within a trilist) generated by the SO function performed by the thread (exclusive of any SOB overflow). If the SO function is not enabled, this field MBZ. This field is ignored if the **SVBI Post Increment** field is set in the GS_3DSTATE, described in the 3D pipeline volumes. Format: U16 Range: [0,1024]

FF_SYNC Writeback Message

(Both GS & Clip): DWord W0.0 of the writeback data contains initial handle information. If **Handle Valid** is clear, the FF unit did not have a handle available to be allocated as the initial handle – the thread must use the URB_WRITE message to obtain the initial handle. Otherwise the Handle ID and URB Return Handle fields are valid and can be used to write the first VUE.

(GS-only): The writeback data contains the SVBI values used as starting write indices by the GS thread. It is the responsibility of the GS thread to perform SOB overflow processing. If the GS thread is not performing StreamOutput and was simply using the writeback to provide GS vertex output synchronization, the return data is to be ignored.

(Clip-only): Dwords W0.1-7 of the writeback data are ignored.

DWord	Bits	Description
W0.7:5	31:0	Reserved
W0.4	31:0	**(GS-only): Streamed Vertex Buffer Index 3** This field represents the value of SVBI[3] that is the starting index for the GS thread. If the thread is not performing StreamOutput, this field is ignored. Format = U32 Range = $[0,2^{27}-1]$
W0.3	31:0	**(GS-only) Streamed Vertex Buffer Index 2**
W0.2	31:0	**(GS-only) Streamed Vertex Buffer Index 1**
W0.1	31:0	**(GS-only) Streamed Vertex Buffer Index 0**
W0.0	31:24	Reserved
	23:16	**Handle ID.** This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. Format: Reserved for Implementation Use

DWord	Bits	Description
	15:12	Reserved
	11:0	**URB Return Handle.** This is the initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the first destination URB entry. Format: U12 256-bit URB Handle Address

URB Messages

This section documents the global aspects of the URB messages. The actual data stored in URB entries differs for each fixed function – refer to *3D Pipeline* and the fixed-function chapters or details on 3D URB data formats and *Media* for media-specific URB data formats.

URB Handles: Unlike prior products where the URB handle contents was not specified for software use, URB handles are now specified as offsets into the URB partition in the L3 cache, in 512-bit units. Thus, kernels can now perform math operations on URB handles.

The **End of Thread** bit in the message descriptor may be set on URB messages only in threads dispatched by the vertex shader (VS), hull shader (HS), domain shader (DS), and geometry shader (GS). The **End of Thread** bit cannot be set on URB_READ* or URB_ATOMIC* messages.

Execution Mask. The low 8 bits of the execution mask on the send instruction determines which DWords from each write data phase are written or which DWords from each read phase are written to the destination GRF register. The execution mask is ignored on URB_ATOMIC* messages, because this is a scalar operation that is always enabled.

Out-of-Bounds Accesses. Reads to addresses outside of the URB region allocated in the L3 cache return 0. Writes to addresses outside of the URB region are dropped and do not modify any URB data.

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
URB Read HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Write HWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Read OWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset +	1, 2

Message Type	Header Required	Shared Local Memory Support	Stateless Support	Address Modes	Vector Width
				offset	
URB Write OWORD	yes	N/A	N/A	handle + URBoffset or handle + URBoffset + offset	1, 2
URB Atomic MOV	yes	N/A	N/A	handle + URBoffset	1
URB Atomic INC	yes	N/A	N/A	handle + URBoffset	1
URB Atomic ADD	yes	N/A	N/A	handle + URBoffset	1

“offset” is in the message payload, and is per-slot.
“handle” is the handle address in the message header.
“URBoffset” is the **Global Offset** field in the URB message descriptor.

Execution Mask

The Execution Mask specified in the ‘send’ instruction determines which DWords within each message register are read/written to the URB.

Message Descriptor

Bit	Description
19	**Header Present** This bit must be 1 for all URB messages.
18	Ignored.
17	**Per Slot offset:** If clear, the slot offset fields in the header are ignored. If set the slot offset fields are added to the global offset to obtain the overall offset. **Programming Notes:** • This bit must be 0 for URB_ATOMIC_* messages.
16	Ignored.
15	**Swizzle Control.** This field specifies which “swizzle” operation is performed on the write data. It indirectly specifies whether one or two handles are valid. 0: URB_NOSWIZZLE

Bit	Description
	The message accesses a single URB entry (using **URB Handle 0**). 1: URB_INTERLEAVED The message accesses two URB entries. The data is interleaved such that the upper DWords (7:4) of each 256-bit unit contain data associated with **URB Handle 1**, and the lower DWords (3:0) contain data associated with **URB Handle 0**. **Programming Notes:** • This bit must be 0 for URB_ATOMIC_* messages.
14:4	**Global Offset.** This field specifies a destination offset (in 256-bit units) from the start of the URB entry(s), as referenced by **URB Handle** *n*, at which the data (if any) is written to or read from. When URB_INTERLEAVED is used, this field provides a 256-bit granular offset applied to both URB entries. If the **Per Slot Offset** bit is set, this offset is added to the per-slot offsets in the header to obtain the overall offset. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. For the URB_ATOMIC* messages, this offset is in 32-bit units instead of 256-bit units. Format = U11 Range = [0, 1023] for URB_*_HWORD messages. Range = [0, 2047] for URB_*_OWORD messages. Range = [0, 2047] for URB_ATOMIC* messages.
3:0	**URB Opcode** 0: URB_WRITE_HWORD 1: URB_WRITE_OWORD 2: URB_READ_HWORD 3: URB_READ_OWORD 4: URB_ATOMIC_MOV 5: URB_ATOMIC_INC 6: URB_ATOMIC_ADD 7: URB_SIMD8_WRITE (see URB_SIMD8_* for descriptor details) 8: URB_SIMD8_READ (see URB_SIMD8_* for descriptor details) 9-15: Reserved

URB_WRITE and URB_READ

The URB_WRITE* and URB_READ* messages share the same header definition. URB_WRITE has additional payload containing the write data, but has no writeback message. URB_READ has no payload beyond the header (message length is always one), but always has a writeback message. URB_WRITE_SIMD4x2 has a single-phase payload with the per-slot offsets followed by the write data, and has no writeback message. URB_READ_SIMD4x2 has a single phase payload containing the per-slot offsets.

Message Header

M0.5[7:0] bits in message header are used for enabling DWs in cull test, at HDC unit by HS kernel, while writing TF data using URB write messages. Cull test is performed on outside TF and HS kernel set the appropriate DW enable, which carry the TF for different domain types. When DW is enabled and if cull test is positive, HS stage will be informed by HDC unit, to cull the HS handle early at HS stage itself.

DWord	Bits	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:17	Ignored
	16	**High OWORD Enable** For URB_READ_OWORD and URB_WRITE_OWORD with NOSWIZZLE indicates whether the 128 bits of the GRF register is used. 0: 1 OWord, read into or written from the low 128 bits of the GRF register. 1: 1 OWord, read into or written from the high 128 bits of the GRF register.
	15	Vertex 1 DATA [3] / Vertex 0 DATA[7] Channel Mask When **Swizzle Control** = URB_INTERLEAVED this bit controls Vertex 1 DATA[3]. When **Swizzle Control** = URB_NOSWIZZLE this bit controls Vertex 0 DATA[7]. This bit is ANDed with the corresponding channel enable to determine the final channel enable. For the URB_READ_OWORD & URB_READ_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be included in the writeback message. For the URB_WRITE_OWORD & URB_WRITE_HWORD messages, when final channel enable is 1 it indicates that Vertex 1 DATA [3] will be written to the surface. 0: Vertex 1 DATA [3] / Vertex 0 DATA[7] channel not included. 1: Vertex DATA [3] / Vertex 0 DATA[7] channel included.
	14	**Vertex 1 DATA [2] Channel Mask**
	13	**Vertex 1 DATA [1] Channel Mask**
	12	**Vertex 1 DATA [0] Channel Mask**
	11	**Vertex 0 DATA [3] Channel Mask**

DWord	Bits	Description
	10	**Vertex 0 DATA [2] Channel Mask**
	9	**Vertex 0 DATA [1] Channel Mask**
	8	**Vertex 0 DATA [0] Channel Mask**
	7:0	**Reserved**
M0.4	31:0	**Slot 1 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 1**, at which the data will be accessed. This field is ignored unless **Swizzle Control** is set to URB_INTERLEAVED. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
M0.3	31:0	**Slot 0 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 256-bit units) from the start of the URB entry, as referenced by **URB Handle 0**, at which the data will be accessed. For the URB_*_OWORD messages, this offset is in 128-bit units instead of 256-bit units. Format = U32 Range = [0, 1023] for URB_*_HWORD messages. The range of the calculated offset must fall within the range [0, 1023] or behavior is undefined. Range = [0, 2047] for URB_*_OWORD messages. The range of the calculated offset must fall within the range [0, 2047] or behavior is undefined.
	31:0	Reserved.
M0.1	31:16	**Handle ID 1.** This ID is assigned by the fixed function unit and links the work in channel 1 to a specific entry within the fixed function unit. This field is ignored unless **Swizzle Control** indicates Interleave mode.
	15:14	Reserved.
	13:0	**URB Handle 1.** This is the URB handle where channel 1's results are to be written or read. This field is ignored unless **Swizzle Control** indicates interleave mode.
M0.0	31:16	**Handle ID 0.** This ID is assigned by the fixed function unit and links the work in channel 0 to a specific entry within the fixed function unit. This field is ignored unless **Swizzle Control** indicates Interleave mode.
	15:14	Reserved.
	13:0	**URB Handle 0.** This is the URB handle where channel 0's results are to be written or read.

URB_WRITE_HWORD Write Data Payload

Programming Restriction: The write data payload can be between 1 and 8 message phases long.

For the URB_WRITE_HWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into consecutive URB locations (no data swizzling applied).

Programming Note	
Context:	URB_NOSWIZZLE
The URB function will use (not ignore) the Channel Enables associated with this message.	

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
M1.7	31:0	**Vertex Data [7]**
M1.6	31:0	**Vertex Data [6]**
M1.5	31:0	**Vertex Data [5]**
M1.4	31:0	**Vertex Data [4]**
M1.3	31:0	**Vertex Data [3]**
M1.2	31:0	**Vertex Data [2]**
M1.1	31:0	**Vertex Data [1]**
M1.0	31:0	**Vertex Data [0]**
M2.7	31:0	**Vertex Data [15]**
M2.6	31:0	**Vertex Data [14]**
M2.5	31:0	**Vertex Data [13]**
M2.4	31:0	**Vertex Data [12]**
M2.3	31:0	**Vertex Data [11]**
M2.2	31:0	**Vertex Data [10]**
M2.1	31:0	**Vertex Data [9]**
M2.0	31:0	**Vertex Data [8]**

DWord	Bit	Description
...		**...**
Mn.7	31:0	**Vertex Data [8(n-1)+7]**
Mn.6	31:0	**Vertex Data [8(n-1)+6]**
Mn.5	31:0	**Vertex Data [8(n-1)+5]**
Mn.4	31:0	**Vertex Data [8(n-1)+4]**
Mn.3	31:0	**Vertex Data [8(n-1)+3]**
Mn.2	31:0	**Vertex Data [8(n-1)+2]**
Mn.1	31:0	**Vertex Data [8(n-1)+1]**
Mn.0	31:0	**Vertex Data [8(n-1)+0]**

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

Programming Note	
Context:	URB_INTERLEAVED
• The URB function will use (not ignore) the Channel Enables associated with this message. • Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.	

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**
M2.7	31:0	**Vertex 1 Data [7]**
M2.6	31:0	**Vertex 1 Data [6]**
M2.5	31:0	**Vertex 1 Data [5]**
M2.4	31:0	**Vertex 1 Data [4]**
M2.3	31:0	**Vertex 0 Data [7]**
M2.2	31:0	**Vertex 0 Data [6]**
M2.1	31:0	**Vertex 0 Data [5]**
M2.0	31:0	**Vertex 0 Data [4]**

DWord	Bit	Description
...		**...**
Mn.7	31:0	**Vertex 1 Data [4(n-1)+3]**
Mn.6	31:0	**Vertex 1 Data [4(n-1)+2]**
Mn.5	31:0	**Vertex 1 Data [4(n-1)+1]**
Mn.4	31:0	**Vertex 1 Data [4(n-1)+0]**
Mn.3	31:0	**Vertex 0 Data [4(n-1)+3]**
Mn.2	31:0	**Vertex 0 Data [4(n-1)+2]**
Mn.1	31:0	**Vertex 0 Data [4(n-1)+1]**
Mn.0	31:0	**Vertex 0 Data [4(n-1)+0]**

URB_READ_HWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

Programming Restriction: The writeback message can be between 1 and 8 message phases long.

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing *n* pairs of 4-DWord vertex elements (where for the example, *n* is >2).

DWord	Bit	Description
W0.7	31:0	**Vertex Data [7]**
W0.6	31:0	**Vertex Data [6]**
W0.5	31:0	**Vertex Data [5]**
W0.4	31:0	**Vertex Data [4]**

DWord	Bit	Description
W0.3	31:0	**Vertex Data [3]**
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**
W0.0	31:0	**Vertex Data [0]**
W1.7	31:0	**Vertex Data [15]**
W1.6	31:0	**Vertex Data [14]**
W1.5	31:0	**Vertex Data [13]**
W1.4	31:0	**Vertex Data [12]**
W1.3	31:0	**Vertex Data [11]**
W1.2	31:0	**Vertex Data [10]**
W1.1	31:0	**Vertex Data [9]**
W1.0	31:0	**Vertex Data [8]**
...		**...**
Wn.7	31:0	**Vertex Data [8n+7]**
Wn.6	31:0	**Vertex Data [8n+6]**
Wn.5	31:0	**Vertex Data [8n+5]**
Wn.4	31:0	**Vertex Data [8n+4]**
Wn.3	31:0	**Vertex Data [8n+3]**
Wn.2	31:0	**Vertex Data [8n+2]**
Wn.1	31:0	**Vertex Data [8n+1]**
Wn.0	31:0	**Vertex Data [8n+0]**

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing *n* 4-DWord vertex elements (n>1).

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**
W0.3	31:0	**Vertex 0 Data [3]**
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**
W1.7	31:0	**Vertex 1 Data [7]**
W1.6	31:0	**Vertex 1 Data [6]**

DWord	Bit	Description
W1.5	31:0	**Vertex 1 Data [5]**
W1.4	31:0	**Vertex 1 Data [4]**
W1.3	31:0	**Vertex 0 Data [7]**
W1.2	31:0	**Vertex 0 Data [6]**
W1.1	31:0	**Vertex 0 Data [5]**
W1.0	31:0	**Vertex 0 Data [4]**
...		**...**
Wn.7	31:0	**Vertex 1 Data [4n+3]**
Wn.6	31:0	**Vertex 1 Data [4n+2]**
Wn.5	31:0	**Vertex 1 Data [4n+1]**
Wn.4	31:0	**Vertex 1 Data [4n+0]**
Wn.3	31:0	**Vertex 0 Data [4n+3]**
Wn.2	31:0	**Vertex 0 Data [4n+2]**
Wn.1	31:0	**Vertex 0 Data [4n+1]**
Wn.0	31:0	**Vertex 0 Data [4n+0]**

URB_WRITE_OWORD Write Data Payload

For the URB_WRITE_OWORD messages, the message payload will be written to the URB entries indicated by the URB return handles in the message header.

Payload	Usage
URB_NOSWIZZLE	The message payload contains data to be written to a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The message payload contains data to be written to two separate URB entries. The payload data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply write data into a single 128-bit URB location (no data swizzling applied).

Programming Note	
Context:	URB_NOSWIZZLE
The URB function will use (not ignore) the Channel Enables associated with this message.	

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
M1.7	31:0	Ignored
M1.6	31:0	Ignored
M1.5	31:0	Ignored
M1.4	31:0	Ignored
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

When URB_NOSWIZZLE is used to write vertex data, the following table shows an example layout of a URB_NOSWIZZLE payload containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
M1.7	31:0	**Vertex 0 Data [3]**
M1.6	31:0	**Vertex 0 Data [2]**
M1.5	31:0	**Vertex 0 Data [1]**
M1.4	31:0	**Vertex 0 Data [0]**
M1.3	31:0	Ignored
M1.2	31:0	Ignored
M1.1	31:0	Ignored
M1.0	31:0	Ignored

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

Programming Note	
Context:	URB_INTERLEAVED
• The URB function will use (not ignore) the Channel Enables associated with this message. • Writes to overlapping addresses of vertex0 and vertex1 will have undefined write ordering.	

DWord	Bit	Description
M1.7	31:0	**Vertex 1 Data [3]**
M1.6	31:0	**Vertex 1 Data [2]**

DWord	Bit	Description
M1.5	31:0	**Vertex 1 Data [1]**
M1.4	31:0	**Vertex 1 Data [0]**
M1.3	31:0	**Vertex 0 Data [3]**
M1.2	31:0	**Vertex 0 Data [2]**
M1.1	31:0	**Vertex 0 Data [1]**
M1.0	31:0	**Vertex 0 Data [0]**

URB_READ_OWORD Writeback Message

For the URB_READ_HWORD messages, the URB entries indicated by the URB handles in the message header are read and returned in the writeback message. The amount of read data returned is determined by the **Response Length** field.

Programming Note	
Context:	URB_READ_OWORD Writeback Message
Response Length must be set to 1.	

While GS threads will read one vertex at a time to the URB, the VS will read two interleaved vertices. The description of the URB read messages will refer to the per-vertex DWords described in the Vertex URB Entry Formats section of the *3D Overview* chapter.

Payload	Usage
URB_NOSWIZZLE	The writeback message contains data read from a single URB entry (e.g., one Vertex URB entry). The **Swizzle Control** field of the message descriptor must be set to 'NoSwizzle'.
URB_INTERLEAVED	The writeback message contains data read from two separate URB entries. The data is provided in a high/low interleaved fashion. The **Swizzle Control** field of the message descriptor must be set to 'Interleave'.

URB_NOSWIZZLE

URB_NOSWIZZLE is used to simply read data into consecutive URB locations (no data interleaving applied).

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 0.

DWord	Bit	Description
W0.7	31:0	Reserved (not written to GRF)
W0.6	31:0	Reserved (not written to GRF)
W0.5	31:0	Reserved (not written to GRF)
W0.4	31:0	Reserved (not written to GRF)
W0.3	31:0	**Vertex Data [3]**

DWord	Bit	Description
W0.2	31:0	**Vertex Data [2]**
W0.1	31:0	**Vertex Data [1]**
W0.0	31:0	**Vertex Data [0]**

When URB_NOSWIZZLE is used to read vertex data, the following table shows an example layout of a URB_NOSWIZZLE writeback message containing one (non-interleaved) vertex containing 4-DWord vertex elements and HIGH OWORD ENABLE is 1.

DWord	Bit	Description
W0.7	31:0	**Vertex Data [3]**
W0.6	31:0	**Vertex Data [2]**
W0.5	31:0	**Vertex Data [1]**
W0.4	31:0	**Vertex Data [0]**
W0.3	31:0	Reserved (not written to GRF)
W0.2	31:0	Reserved (not written to GRF)
W0.1	31:0	Reserved (not written to GRF)
W0.0	31:0	Reserved (not written to GRF)

URB_INTERLEAVED

The following table shows an example layout of a URB_INTERLEAVED payload containing two interleaved vertices, each containing 4-DWord vertex elements.

DWord	Bit	Description
W0.7	31:0	**Vertex 1 Data [3]**
W0.6	31:0	**Vertex 1 Data [2]**
W0.5	31:0	**Vertex 1 Data [1]**
W0.4	31:0	**Vertex 1 Data [0]**
W0.3	31:0	**Vertex 0 Data [3]**
W0.2	31:0	**Vertex 0 Data [2]**
W0.1	31:0	**Vertex 0 Data [1]**
W0.0	31:0	**Vertex 0 Data [0]**

URB_ATOMIC

The URB_ATOMIC messages implement atomic operations on a single DWord in the URB. The location of the DWord within the URB is specified by the single URB handle and the **Global Offset** field in the message descriptor, which for these messages is a DWord offset from the URB handle. The DWord selected is operated on according to the following table:

URB Opcode	new_dst	ret
URB_ATOMIC_MOV	0	none
URB_ATOMIC_INC	old_dst + 1	old_dst
URB_ATOMIC_ADD	old_dst + src0	old_dst

The previous contents of the DWord are returned in the destination register for operations that update the DWord value, such as URB_ATOMIC_INC. The URB_ATOMIC_MOV opcode does not return data (response length must be zero).

The URB_ATOMIC* messages consist only of the header. A single URB handle is specified.

Message Header

DWord	Bits	Description
M0.7	31:0	**Reserved**
M0.6	31:0	**Reserved**
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	**Source0 Data** Specifies the source 0 data for the atomic operation. This field is ignored for the URB_ATOMIC_INC message. Format = U32
M0.1	31:0	Ignored
M0.0	31:16	Ignored
	15:0	**URB Handle.** The URB handle to access.

Writeback Message

A writeback message is only returned for URB atomic operations that update the DWord value, such as URB_ATOMIC_INC. Only the low 32 bits of the destination GRF register are overwritten with the return data.

DWord	Bits	Description
W0.7:1		Reserved (not written to GRF)
W0.0	31:0	**Return Data** Specifies the value of the return data for the atomic operation. Format = U32

URB_SIMD8_Write and URB_SIMD8_Read

Programming Note	
Context:	URB_SIMD8_Write and URB_SIMD8_Read
The constant, sampler, and render caches are always non-coherent.	

Message Descriptor

Bit	Description
19	**Header Present** This bit must be set to **one** for all URB messages.
18	Ignored
17	**Per Slot offset Present:** If clear, then slot offset message phase is absent. If set then slot offset message phase is present and the per slot offsets are added to the global offset to obtain the overall offset.
16	**Ignored**
15	**Channel Mask Present:** If clear then the channel Mask Message phase is not present. If set then the channel mask message phase is present and will be used to mask data elements read or written.
14:4	**Global Offset.** This field specifies a destination offset (in 128-bit units) from the start of the URB entry(s), as referenced by **URB Handle** *n*, at which the data (if any) will be written to or read from. If the **Per Slot Offset** bit is set, this offset is added to the per-slot offsets in the header to obtain the overall offset. Format = U11 Range = [0, 2047]
3:0	**URB Opcode** 0: URB_WRITE_HWORD 1: URB_WRITE_OWORD 2: URB_READ_HWORD 3: URB_READ_OWORD 4: URB_ATOMIC_MOV 5: URB_ATOMIC_INC 6: URB_ATOMIC_ADD

Bit	Description
	7: URB_SIMD8_WRITE 8: URB_SIMD8_READ 9-15: Reserved

Message Header

DWord	Bits	Description
M0.7	31:16	**Handle ID 7.** This ID is assigned by the fixed function unit and links the work in channel 7 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 7.** This is the URB handle where channel 7's results are written or read.
M0.6	31:16	**Handle ID 6.** This ID is assigned by the fixed function unit and links the work in channel 6 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 6.** This is the URB handle where channel 6's results are written or read.
M0.5	31:16	**Handle ID 5.** This ID is assigned by the fixed function unit and links the work in channel 5 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 5.** This is the URB handle where channel 5's results are written or read.
M0.4	31:16	**Handle ID 4.** This ID is assigned by the fixed function unit and links the work in channel 4 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 4.** This is the URB handle where channel 4's results are written or read.
M0.3	31:16	**Handle ID 3.** This ID is assigned by the fixed function unit and links the work in channel 3 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 3.** This is the URB handle where channel 3's results are written or read.
M0.2	31:16	**Handle ID 2.** This ID is assigned by the fixed function unit and links the work in channel 2 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 2.** This is the URB handle where channel 2's results are written or read.
M0.1	31:16	**Handle ID 1.** This ID is assigned by the fixed function unit and links the work in channel 1 to a specific entry within the fixed function unit.
	15:14	Reserved
	13:0	**URB Handle 1.** This is the URB handle where channel 1's results are written or read.
M0.0	31:16	**Handle ID 0.** This ID is assigned by the fixed function unit and links the work in channel 0 to a specific entry within the fixed function unit.
	15:14	Reserved

DWord	Bits	Description
	13:0	**URB Handle 0.** This is the URB handle where channel 0's results are written or read.

Per Slot Offset Message Phase

When the **Per Slot offset Present** bit in the descriptor is set then the Per slot offset message phase is sent by the EUs. The per slot message phase occurs immediately after the header.

DWord	Bit	Description
M1.7	31:0	**Slot 7 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 7**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.6	31:0	**Slot 6 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 6**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.5	31:0	**Slot 5 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 5**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.4	31:0	**Slot 4 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 4**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.3	31:0	**Slot 3 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 3**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.2	31:0	**Slot 2 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 2**, at

DWord	Bit	Description
		which the data will be accessed. Format = U32 Range = [0, 2047]
M1.1	31:0	**Slot 1 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 1**, at which the data will be accessed. Format = U32 Range = [0, 2047]
M1.0	31:0	**Slot 0 Offset.** This field, after adding to the **Global Offset** field in the message descriptor, specifies the offset (in 128-bit units) from the start of the URB entry, as referenced by **URB Handle 0**, at which the data will be accessed. Format = U32 Range = [0, 2047]

Channel Mask Message Phase

When the **Channel Mask Present** bit in the descriptor is set then the channel mask message phase is sent by the EUs. The channel mask message phase occurs after the per slot message phase if the per slot message phase exists else it occurs after the header.

DWord	Bit	Description
M2.7	31:24	**Reserved.**
	23	Vertex 7 DATA [7] Channel Mask This bit is ANDed with the corresponding channel enable to determine the final channel enable. For the URB_SIMD8_READ messages, when final channel enable is 1 it indicates that Vertex 7 DATA [7] will be included in the writeback message. For the URB_SIMD8_WRITE messages, when final channel enable is 1 it indicates that Vertex 7 DATA [7] will be written to the surface. 0: Vertex 7 DATA [7] channel not included 1: Vertex 7 DATA [7] channel included
	22	**Vertex 7 DATA [6] Channel Mask**
	21	**Vertex 7 DATA [5] Channel Mask**
	20	**Vertex 7 DATA [4] Channel Mask**
	19	**Vertex 7 DATA [3] Channel Mask**
	18	**Vertex 7 DATA [2] Channel Mask**
	17	**Vertex 7 DATA [1] Channel Mask**
	16	**Vertex 7 DATA [0] Channel Mask**

DWord	Bit	Description
	15:0	**Reserved.**
M2.6	31:24	**Reserved.**
	23:16	**Vertex 6 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.5	31:24	**Reserved.**
	23:16	**Vertex 5 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.4	31:24	**Reserved.**
	23:16	**Vertex 4 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.3	31:24	**Reserved.**
	23:16	**Vertex 3 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.2	31:24	**Reserved.**
	23:16	**Vertex 2 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.1	31:24	**Reserved.**
	23:16	**Vertex 1 DATA [7:0] Channel Mask**
	15:0	**Reserved.**
M2.0	31:24	**Reserved.**
	23:16	**Vertex 0 DATA [7:0] Channel Mask**
	15:0	**Reserved.**

Write Data Payload

The write data payload can be between 1 and 8 message phases long.

DWord	Bit	Description
M3.7	31:0	**Vertex 7 DATA [0]**
M3.6	31:0	**Vertex 6 DATA [0]**
M3.5	31:0	**Vertex 5 DATA [0]**
M3.4	31:0	**Vertex 4 DATA [0]**
M3.3	31:0	**Vertex 3 DATA [0]**
M3.2	31:0	**Vertex 2 DATA [0]**
M3.1	31:0	**Vertex 1 DATA [0]**
M3.0	31:0	**Vertex 0 DATA [0]**
...	...	**...**
M10.7	31:0	**Vertex 7 DATA [7]**

DWord	Bit	Description
M10.6	31:0	**Vertex 6 DATA [7]**
M10.5	31:0	**Vertex 5 DATA [7]**
M10.4	31:0	**Vertex 4 DATA [7]**
M10.3	31:0	**Vertex 3 DATA [7]**
M10.2	31:0	**Vertex 2 DATA [7]**
M10.1	31:0	**Vertex 1 DATA [7]**
M10.0	31:0	**Vertex 0 DATA [7]**

Writeback Message

The writeback message can be between 1 and 8 message phases long.

DWord	Bit	Description
M0.7	31:0	**Vertex 7 DATA [0]**
M0.6	31:0	**Vertex 6 DATA [0]**
M0.5	31:0	**Vertex 5 DATA [0]**
M0.4	31:0	**Vertex 4 DATA [0]**
M0.3	31:0	**Vertex 3 DATA [0]**
M0.2	31:0	**Vertex 2 DATA [0]**
M0.1	31:0	**Vertex 1 DATA [0]**
M0.0	31:0	**Vertex 0 DATA [0]**
...	...	**...**
M7.7	31:0	**Vertex 7 DATA [7]**
M7.6	31:0	**Vertex 6 DATA [7]**
M7.5	31:0	**Vertex 5 DATA [7]**
M7.4	31:0	**Vertex 4 DATA [7]**
M7.3	31:0	**Vertex 3 DATA [7]**
M7.2	31:0	**Vertex 2 DATA [7]**
M7.1	31:0	**Vertex 1 DATA [7]**
M7.0	31:0	**Vertex 0 DATA [7]**

Message Gateway

The Message Gateway shared function provides a mechanism for active thread-to-thread communication. Such thread-to-thread communication is based on direct register access. One thread, a *requester thread*, is capable of writing into the GRF register space of another thread, a *recipient thread*. Such direct register access between two threads in a multi-processor environment some time is referred to as *remote register access*. Remote register access may include read or write. The architecture supports *remote register write*, but not remote register read (natively). Message Gateway facilitates such remote register write via message passing. The requester thread sends a message to Message Gateway requesting a write to the recipient thread's GRF register space. Message Gateway sends a writeback message to the recipient thread to complete the register write on behalf of the requester. The requester thread and the recipient thread may be on the same EU or on different EUs.

Please see Thread Spawn Message Section of Media Chapter for child thread termination using Message Gateway messages with EOT bit set.

When Bypass Gateway Control is set to 1, the commands OpenGateway and CloseGateway are no longer used, the gateway parameters are taking the default values as the following:

- **RegBase** = 0
- **Gateway Size** check and **Key** check are bypassed.
- **Gateway Open** (an internal signal that is used to be set by OpenGateway message) check is bypassed

A separate Gateway exists per half-slice in the architecture. For ForwardMsg this is handled transparently, but barriers can only be accessed by threads in the local half-slice. This means that all threads that access a shared barrier need to use the half-slice select in GPGPU_OBJECT and MEDIA_OBJECT to stay on a single half-slice. GPGPU_WALKER handles this automatically.

Messages

Message Gateway supports such thread-to-thread communication with the following messages:

- **OpenGateway:** Opens a gateway for a requester thread. Once a thread successfully opens its gateway, it can be a recipient thread to receive remote register write.
- **CloseGateway:** Closes the gateway for a requester thread. Once a thread successfully closes its gateway, Message Gateway blocks any future remote register writes to this thread.
- **ForwardMsg:** Forwards a formatted message (remote register write) from a requester thread to a recipient thread.
- **GetTimeStamp:** Reads absolute and relative timestamps for a requester thread.
- **BarrierMsg:** A set of threads sends this message to the Gateway. When all threads in a group have sent the message, a reply (both a register write and an N0 notification) is sent to each member of the group.
- **UpdateGatewayState:** Updates the internal state of the Message Gateway.

  One example usage is to allow a control thread to change Barrier Byte to convey dynamic state information. This may be used to support interrupt when persistent compute/worker threads are synchronized using Barrier.

MMIO Read/Write: allows a message to read or write an MMIO register. The MEDIA_VFE_STATE command has a field which limits the accesses for security.

Message Descriptor

The following message descriptor applies to all messages supported by Message Gateway.

Bits	Description
19	**Header Present.** This bit must be 0 for all Message Gateway messages.
18:17	Ignored.
16:15	**Notify.** Send Notification Signal. This is a two-bit field indicating which notify event is sent. 00b: No notify. 01b: Increment recipient thread's N0 notification counter. 10b: Increment recepient thread's N2 notification counter. 11b: Reserved. This field is only valid for a ForwardMsg message. It is ignored for other messages. The BarrierMsg message always increments the N0 notification counter.
14	**AckReq.** Acknowledgment Required. When this bit is set, an acknowledgment return message is required. Message Gateway sends a writeback message containing the error code to the requester thread using the post destination register address. When this bit is 0, no writeback message is sent to the requesting

	thread by Message Gateway, even if an error occurs. This field is valid for OpenGateway, CloseGateway, ForwardMsg, and BarrierMsg messages. When this bit is 1, post destination register must be valid and the response length must be 1. When this bit is 0, post destination register must be *null* and the response length must be 0. This bit cannot be set when EOT is set; otherwise, hardware behavior is undefined. 0: No Acknowledgement is required. 1: Acknowledgement is required.
13:3	Reserved: MBZ
2:0	**SubFuncID.** Identify the supported sub-functions by Message Gateway. Encodings are: 000b: **OpenGateway.** Open the gateway for the requester thread. 001b: **CloseGateway.** Close the gateway for the requester thread. 010b: **ForwardMsg.** Forward the formatted message to the recipient thread with the given offset from the recipient's register base. 011b: **GetTimeStamp**. Read absolute and relative timestamps. 100b: **BarrierMsg.** Record an additional thread reaching the barrier. 101b: **UpdateGatewayState.** Update the barrier byte for a barrier. **Value** / **Description** 110b: Reserved. 110b: **MMIO Read/Write.** 111b: Reserved.

OpenGateway Message

The OpenGateway message opens a communication channel between the requesting thread and other threads. It specifies a key for other threads to access its gateway, as well as the GRF register range allowed to be written. The message consists of a single 256-bit message payload.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completion of the OpenGateway function. Only the least significant DWord in the post destination register is overwritten.

If the EOT is set for this message, Message Gateway ignores this message; instead, it closes the gateway for the requesting thread regardless of the previous state of the gateway.

It is software's policy to determine how to generate the key.

The BarrierMsg command does not use an OpenGateway message.

Message Payload

DWord	Bits	Description
M0.5	31:29	Reserved: MBZ
	28:21	**RegBase:** The register base address to be stored in the Message Gateway. It is used to compute the destination GRF register address from the offset field in ForwardMsg. RegBase contains 256-bit GRF aligned register address. Note 1: This field aligns with bits [28:21] of the Offset field of the message payload for ForwardMsg. Note 2: The most significant bit of this field must be zero. Format = U8 Range = [0,127]
	20:11	**Reserved: MBZ**
	10:8	**Gateway Size:** The range limit for messages through the Message Gateway. 000b: 1 GRF Register 001b: 2 GRF Registers 010b: 4 GRF Registers 011b: 8 GRF Registers 100b: 16 GRF Registers 101b: 32 GRF Registers 110b: 64 GRF Registers 111b: 128 GRF Registers
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway. This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:16	Reserved: MBZ
	15:0	Reserved: MBZ.
M0.3:0	31:0	Ignored

Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bits	Description
W0.7:1	31:0	Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID.** The message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000b: **Successful**. No Error (Normal). 101b: **Opcode Error.** Attempt to send a message which is not either open/close/forward. Other codes: Reserved.

CloseGateway Message

The CloseGateway message closes a communication channel for the requesting thread that was previously opened with OpenGateway. Each thread is allowed to have only one open gateway at a time, thus no additional information in the message payload is required to close the gateway. The message consists of a single 256-bit message payload.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requesting thread after completing the CloseGateway function. Only the least significant DWord in the post destination register is overwritten.

Programming Note	
Context:	CloseGateway Message
The BarrierMsg command does not use a CloseGateway message.	

Message Payload

DWord	Bit	Description
M0.7:6		Ignored
M0.5	31:8	Ignored
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4:0		Ignored

Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bit	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID:** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful*. No Error (Normal) 101: *Opcode Error*. Attempt to send a message which is not either open/close/forward other codes: Reserved

ForwardMsg Message

The ForwardMsg message gives the ability for a requester thread to write a **data segment** in the form of a byte, a dword, 2 dwords, or 4 dwords to a GRF register in a recipient thread. The message consists of a single 256-bit message payload, which contains the specially formatted data segment.

The ForwardMsg message utilizes a communication channel previously opened by the recipient thread. The recipient thread has communicated its EUID, TID, and key to the requester thread previously via some other mechanism. Generally, this is done through the thread spawn message from parent to child thread, allowing each child (requester) to then communicate with its parent through a gateway opened by the parent (recipient). The child could then use ForwardMsg message to communicate its own EUID, TID, and key back to the parent to enable bi-directional communication after opening its own gateway.

If the AckReq bit is set, a single 256-bit payload writeback message is sent back to the requester thread after completion of the ForwardMsg function. Only the least significant DWord in the post destination register is overwritten.

If the Notify bit in the message descriptor is set, a 'notification' is sent to the recipient thread in order to increment the recipient thread's notification counter. This allows multiple messages to be sent to the recipient without waking up the recipient thread. The last message, having this bit set, will then wake up the recipient thread.

Message Payload

DWord	Bits	Description
M0.5	31:29	Reserved: MBZ
	28:16	**Offset:** It provides the destination register position in the recipient thread GRF register space as the offset from the RegBase stored in the recipient thread's gateway entry. The offset is in unit of byte, such that bits [28:21] is the 256-bit aligned register offset and bits [4:0] is the sub-register offset. The sub-register offset must be aligned to the Length field in bits [10:8]. The subfields of Offset are further illustrated as the following. Offset[28:21]: Register offset from the gateway base (Range [0, 127]: bit 12 MBZ) Offset[20:18]: DW offset Offset[17:16]: Byte offset (must be 00b for all DW length cases) **Programming Restriction:** R0 cannot be used as destination GRF register for ForwardMsg. NULL register is also not allowed as destination.
	15:11	Reserved: MBZ
	10:8	**Length:** The length of the data segment. 000: 1 byte 001: 1 word 010: 1 DWord 011: 2 DWords 100: 4 DWords 101-111: Reserved
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway. This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:30	Ignored
	31:30	**SliceID:** The SliceID identifies the slice of the message destination EU. The behavior of sending a message to a non-existent or disabled slice is undefined.
	29:28	**SubSliceID:** The sub-slice ID identifies the sub-slice containing the message destination EU. The behavior of sending a message to a non-existent or disabled slice is undefined.
	27:24	**EUID:** The Execution Unit ID as part of the Recipient field is used to identify the recipient thread to which the message is forwarded.

DWord	Bits	Description
	23:19	Ignored
	18:16	**TID:** The Thread ID as part of the Recipient field is used to identify the recipient thread to which the message is forwarded.
	15:0	**Key:** The key to match with the one stored in the recipient thread's entry in Message Gateway. **CHV, BSW:** Ignored.
	28:26	**SliceID:** The Slice ID as part of the Recipient field is used to identify the slice containing the EU to which the message is forwarded.
	25:24	**SubSliceID:** The sub-slice ID identifies the sub-slice containing the message destination EU. The behavior of sending a message to a non-existent or disabled slice is undefined.
	23:20	**EUID:** The Execution Unit ID as part of the Recipient field is used to identify the recipient thread to which the message is forwarded.
	19	Ignored
	18:16	**TID:** The Thread ID as part of the Recipient field is used to identify the recipient thread to which the message is forwarded.
	15:0	Ignored
M0.3	31:0	**Data Segment DWord 3:** Valid only for the 4-DWord data segment length.
M0.2	31:0	**Data Segment DWord 2:** Valid only for the 4-DWord data segment length.
M0.1	31:0	**Data Segment Dword 1:** Valid only for the 2- and 4-DWord data segment lengths.
M0.0	31:24	**Data Segment Byte 0:** The same byte must be copied to all four positions within this DWord. Valid only for the 1-Byte data segment length. **Data Segment Dword 0:** Valid only for the 1-, 2-, and 4-DWord data segment lengths.
	23:16	Data Segment Byte 0
	15:8	Data Segment Byte 0
	7:0	Data Segment Byte 0

Writeback Message to Requester Thread

The writeback message is only sent if the **AckReq** bit in the message descriptor is set.

DWord	Bits	Description
W0.7:1	31:0	Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID.** The message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000b: **Successful**. No Error (Normal). 001b: Reserved. 010b: **Gateway Closed**. Attempt to send a message through a closed gateway. 011b: Reserved. 100b: Reserved. 101b: **Opcode Error.** Attempt to send a message which is not either open/close/forward. 110b: **Invalid Message Size.** Attempt to forward a message with length greater than 4 DWords. 111b: Reserved.

Writeback Message to Recipient Thread

This message contains the byte or dwords data segment indicated in the message written to the GRF register offset indicated. Only the byte/dword(s) will be enabled, all other data in the GRF register is untouched.

GetTimeStamp Message

The GetTimeStamp message gives the ability for a requester thread to read the timestamps back from the message gateway. The message consists of a single 256-bit message payload.

AbsoluteTimeLap is based on an absolute wall clock in unit of nSec/uSec that is independent of context switch or GPU frequency adjustment. Message Gateway shares the same GPU timestamp. Details can be found in the TIMESTAMP register section in *vol1c Memory Interface and Command Stream*.

RelativeTimeLap is based on a relative time count that is counting the GPU clocks for the context. The relative time count is saved/restored during context switch.

Message Payload

DWord	Bits	Description
M0.5	31	**Return to High GRF**: 0: The return 128-bit data goes to the first half of the destination GRF register. 1: The return 128-bit data goes to the second half of the destination GRF register.
	30:8	Reserved: MBZ
	7:0	**Dispatch ID:** This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion. This field is ignored by Message Gateway. This field is only required for a thread that is created by a fixed function (therefore, not a child thread) and EOT bit is set for the message.
M0.4	31:0	Ignored
M0.3	31:0	Ignored
M0.2	31:0	Ignored
M0.1	31:0	Ignored
M0.0	31:0	Ignored

Writeback Message to Requester Thread

As the writeback message is only sent if the **AckReq** bit in the message descriptor is set, **AckReq** bit must be set for this message.

Only half of the destination GRF register is updated (via write-enables). The other half of the register is not changed. This is determined by the **Return to High GRF** control field.

Writeback Message if Return to High GRF is set to 0:

DWord	Bit	Description
W0.7:4		Reserved (not overwritten)
W0.3	31:0	**RelativeTimeLapHigh:** This field returns the MSBs of time lap for the relative clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.2	31:20	**RelativeTimeLapLow:** This field returns the LSBs of time lap for the relative clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ
W0.1	31:0	**AbsoluteTimeLapHigh:** This field returns the MSBs of time lap for the absolute clock since the previous reset. This field represents 1.024 uSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
W0.0	31:20	**AbsoluteTimeLapLow:** This field returns the LSBs of time lap for the absolute clock since the previous reset. This field represents 1/4 nSec increment of the time stamp. Hardware handles the wraparound (over 64 bit boundary) of the timestamp. Format: U12
	19:0	Reserved : MBZ

Writeback Message if Return to High GRF is set to 1:

DWord	Bit	Description
W0.7	31:0	**RelativeTimeLapHigh**
W0.6	31:20	**RelativeTimeLapLow**
	19:0	Reserved : MBZ
W0.5	31:0	**AbsoluteTimeLapHigh**
W0.4	31:20	**AbsoluteTimeLapLow**
	19:0	Reserved : MBZ
W0.3:0		Reserved : MBZ

BarrierMsg Message

The BarrierMsg message gives the ability for multiple threads to synchronize their progress. This is useful when there are data shared between threads. The message consists of a single 256-bit message payload.

Upon receiving one such message, Message Gateway increments the Barrier counter and marks the Barrier requester thread. There is no immediate response from the Message Gateway when the incremented counter is not equal to the terminating thread count. When the incremented counter value does equal the **Barrier Thread Count**, Message Gateway sends a response back to all the Barrier requester threads and resets the Barrier count to zero.

Programming Note	
Context:	BarrierMsg Message
The Message Gateway assumes that the barrier ID sent in barrier message payload is valid and was allocated by TSG. In the event of a programming error specifying an invalid barrier ID, the results are undefined, and may cause the Message Gateway to stop responding to barrier messages for any thread that it services.	

Message Payload

DWord	Bits	Description
M0.5	31:0	**Ignored**
M0.4	31:0	**Ignored**
M0.3	31:0	**Ignored**
M0.2	31	**Ignored**
	30	**Ignored**
	29:28	**Ignored**
	27:24	**BarrierID.** This field indicates which one from the 16 Barrier States is updated. Format: U4 Note: This field location matches with that of R0 header.
	23:16	**Ignored**
	15	**Barrier Count Enable.** Allows the message to reprogram the terminating barrier count. If set, the stored value of the terminating barrier count is set to the value of Barrier Count field (below), and used for this barrier operation. If clear, the stored value of the terminating barrier count is not modified and the stored value is used for this barrier operation. **Programming Note:** This control is intended only for Hull Shader threads. Do not use this control if the barrier is linked with other barriers in other subslices Format: Enable
	14:9	**Barrier Count.** If Barrier Count Enable is set, this field specifies the terminating barrier count. Otherwise this field is ignored. All threads that belong to a single barrier must deliver the same value for this field for a particular barrier iteration.
	8:0	**Ignored**
M0.1	31:0	**Ignored**
M0.0	31:4	**Ignored**

Writeback Message to Requester Thread

The writeback message is sent only if the **AckReq** bit in the message descriptor is set.

DWord	Bits	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:20	Reserved
	19:16	**Shared Function ID.** Contains the message gateway's shared function ID.
	15:3	Reserved
	2:0	**Error Code** 000: *Successful.* No Error (Normal). 001: *Error (Barrier is inactive) [CHV, BSW]* Other encodings are reserved.

Broadcast Writeback Message

Description
When the count for a Barrier reaches Barrier.Count, the Message Gateway sends the notification bit N0 to each EU/Thread that reached the barrier. A Barrier Return Byte is not sent.

DWord	Bits	Description
W0.7:1		Reserved (not overwritten)
W0.0	31:16	Reserved (not overwritten)
	15:8	Reserved (not overwritten)
	7:0	Reserved (not overwritten)

MMIOReadWrite Message

MMIO read/write is not allowed to registers that are associated with a particular slice.

Message Payload

DWord	Bits	Description
M0.5	31:0	Ignored
M0.4	31:0	Ignored
M0.3	31:1	Ignored
	0	**MMIO R/W:** 0 – MMIO Read – A response is sent to the EU with read data. 1 – MMIO Write – No response is sent to EU (unless acknowledge is requested in sideband).
M0.2	31:28	Ignored
	22:0	**MMIO Address:** The MMIO Byte address to be accessed. The bottom 2 bits must be zero.
M0.1	31:0	Ignored
M0.0	31:0	**MMIO Write Data** (Only if MMIO R/W = 1, otherwise ignored).

Writeback Message to Requester Thread (MMIO Read Only)

DWord	Bit	Description
R0.7	31:0	Ignored
R0.6	31:0	Ignored
R0.5	31:0	Ignored
R0.4	31:0	Ignored
R0.3	31:0	Ignored
R0.2	31:0	Ignored
R0.1	31:0	Ignored
R0.0	31:0	**MMIO Read Data**

Media Sampler

This section describes the functionality of the Media Sampler.

Shared Functions – Video Motion Estimation

The Video Motion Estimation (VME) engine is a shared function that provides motion estimation services. It includes motion estimation for various block sizes and also standard specific operations such as

- Motion estimation and mode decision for AVC
- Intra prediction and mode decision for AVC
- Motion estimation and mode decision for MPEG2
- Motion estimation and mode decision for VC1

The motion estimation engine may also be used for other coding standards or other video processing applications.

Theory of Operation

VME performs a sequence of operations to find the best mode for a given macroblock. Each operation step can be enabled/disabled through the control of the income message. Early termination, skipping of subsequent operation steps, is also supported when certain search criteria are met.

VME contains the following operation steps:

1. Skip check
2. IME: Integer motion estimation
3. FME: Fractional motion estimation
4. BME: Bidirectional motion estimation
5. IPE: Intra prediction estimation (AVC only)

Shape Decision

As a terminology, we call sub-block shapes: 8x4, 4x8, and 4x4 *minor shapes* (corresponding to *sub-partitions* of 8x8 sub-macroblock), and 16x16, 16x8, 8x16, and 8x8 *major shapes* (corresponding to *sub-macroblocks* of a 16x16 macroblock)**.**

If the maximal allowed number of motion vectors **MaxNumMVs** (**MaxNumMVs** = **MaxNumMVsMinusOne** + 1) is less than 4, we will set minor MV flag off: **MinorMVsFlag** = 0, *i.e.* no minor motion vectors will be generated.

The reason of having this parameter **MaxNumMVs** is due to high level AVC conformance restrictions for certain profiles: *the total number of motion vectors of any two consecutive macroblocks not exceeding 16 (or 32)*. The mechanism here allows a reasonable degree of user control. In disable cases, **MaxNumMVs** should be set to 32.

In the coding process of VME, the shape decision is done in multiple locations:

1. After IME and before FME, intermediate shape decision is performed to reduce the FME searching candidates
2. After FME and before BME, existing shape decision is revised among the remaining candidates and to see if there is further reduction.
3. Final shape decision is done after BME.

Partition decision before BME uses unidirectional motion vector count to meet **MaxNumMVs** requirement. Adding BME for the partition candidates may exceed **MaxNumMVs**. As BME is performed on a block by block basis using the block order for a given partition, BME step for a given block is skipped and the best unidirectional motion vectors are used for the block if the overall motion vector count exceeds **MaxNumMVs** when that particular block is switched to bidirectional. The process continues to the last block of the partition.

Note: This is a sub-optimal solution to simplify the hardware implementation. For some cases, bidirectional modes with larger sub-partitions might be better than unidirectional modes with finer sub-partitions.

The VME implementation has the following restriction: Multiple partition candidates are only enabled if **PartCandidateEn** is set. And this only applies to source block of size 16x16.

If **PartCandidateEn** is not set, only the best partition is kept in state 1 (after IME) above and carried through FME and BME. In other words, FME if enabled only operates on one partition candidate, and BME if enabled only operates on one partition candidate. Bidirectional mode check only applies to the partition candidates that meet the bidirectional restriction provided by **BiSubMbPartMask**. For example, if a minor partition determined based on best unidirectional cost function is not 8x8 but one of 4x8, 8x4 or 4x4, VME skips the bidirectional mode check.

If **PartCandidateEn** is set, up to two sets of candidates are maintained by VME hardware, if the second best partition candidate is within **PartToleranceThrhd** from the best one. The second best partition is selected only from the two major partition candidates based on the unidirectional motion vector count, subject to that the major partition is enabled:

- 1MV: The 16x16 partition
- 4MV: The 4x(8x8) partition with no minor shape

The following partitions are not supported as alternative partition.

- 2MV: The best of 2x(16x8) and 2x(8x16) partitions
- More than 4MV: The best of all 4x(8x8) partitions with at least one 8x8 having minor shape of 8x4, 4x8 or 4x4

Minor Shape Decision Prior to FME

If any minor shapes are selected, we decide the best minor first.

For each 8x8 sub-block, before performing bidirectional, we reduce code candidates to no more than three based on the best unidirectional motion search results (best of the forward and backward):

0)One MV, i.e. the best in shape of 8x8.

1)Up to two MVs, i.e. the best in shapes 8x8, 8x4, or 4x8. And

2)Up to four MVs, i.e. the best for the sub-block 8x8.

Now for the first and the second sub-blocks, we can merge them into up to six candidates of 2, 3, 4, 5, 6, and 8 possible motion vectors.

Do the same to the third and the fourth sub-blocks; we have similarly up to six candidates.

Now we further combine these two groups, and find the best solution under the constraint of not exceeding the number of motion vectors more than **MaxNumMVs** (see pseudo-code below for detail).

Consequently, we have the best combined 8x8 solutions with **N** motion vectors for some **N** less or equal to **MaxNumMVs**.

Assume `distA[k][s]` is the cost-adjusted distortion of the best forward or backward motion vector mix of the *k*-th 8x8 sub-block of the sub-shape *s*, where *s*=0, 1, 2, and 3 represent shape partitioning 8x8, 8x4, 4x8, and 4x4 respectively. Assume `distA[k][s]` is the bidirectional one of the corresponding bus-block and sub-shape. And assume some large number, say 128x16=2048 is assigned to the variable, if there were no valid corresponding codes. Hence, the following pseudo-code explains the code selection algorithm.

Let's first explain the case where **MaxNumMVs** is disabled, i.e. **MaxNumMVs**>=16:

```
void SelectBestCombinedMinors(
    short *distA,
    short *MinorShape,
    short *MinorDisto)
{
    short s[4], d[4];
    s = ShapeList;
    d = DistoList;
    for ( int k=0; k<4; k++ ) {
        s[k] = 0;
            d[k] = distA[k][0];
        if ( distA[k][1])<d[k] ) { d[k] = distA[k][1]; s[k] = 1; }
        if ( distA[k][2])<d[k] ) { d[k] = distA[k][3]; s[k] = 2; }
        if ( distA[k][3])<d[k] ) { d[k] = distA[k][3]; s[k] = 3; }
    }
    * MinorDisto = d[0] + d[1] + d[2] + d[3];
    * MinorShape = s[0] | (s[1]«2) | (s[2]«4) | ({s[3]«6};
}
```

Now for the case of using **MaxNumMVs** control:

```
void SelectBestCombinedMinors(
    short *distA,
    int   MaxNumMVs,
    short *MinorShape,
    short *MinorDisto)
{
    int k, n;
    short dist, best0 = 0, best1 = 0;
    if ( MaxNumMVs < 4 ) {  // We reset other parameters.
        switch ( MaxNumMvs ) {
            case 0:
                DoIntraInter &= (~DO_INTER);  // Not do Inter
```

```
                break;

            case 1:
                ShapeMask |= (NO_16X8 | NO_8X16);
                BidirMask |= NO_16X16;
                break;

            case 2:
            case 3:
                ShapeMask |= (NO_8X8 | NO_8X4 | NO_4X8 | NO_4X4);
                BidirMask |= (NO_16X8 | NO_8X16);
                break;
        }
    }
    if ( MaxNumMVs >= 16 ) {  // It should use unrestricted code selection.
        SelectBestCombinedMinors(DistA,MinorShape,MinorDisto);
        return;

    }
    short *s, ShapeList[18];
    short *d, DistoList[18];
    s = ShapeList;
    d = DistoList;
    for ( k=0; k<4; k++ ){
        s[0] = 0;  // 1 mv
        d[0] = distA[k][0];
        s[4] = (distA[k][2] < distA[k][1]) + 1;  // 2 mvs
        d[4] = distA[k][s[1]];
        s[8] = 3;  // 4 mvs
        d[8] = distA[k][3];
        s ++, d ++;
    }
    // Merge two:
    s = ShapeList;
    d = DistoList;
    for ( k=0; k<2; k++ ) {
        s[16] = 0x33;  // 8 mvs
        d[16] = d[8] + d[10];

        s[12] = (d[4] + d[10] < d[6] + d[8]) ? (s[4] | 0x30) : (0x03 | (s[6] « 4));  // 6
mvs
        d[12] = (d[4] + d[10] < d[6] + d[8]) ? (d[4] + d[10]) < (d[6] + d[8]);

        s[10] = (d[0] + d[10] < d[8] + d[2]) ? 0x30 : 0x03;  // 5 mvs
        d[10] = (d[0] + d[10] < d[8] + d[2]) ? (d[0] + d[10]) < (d[8] + d[2]);

        s[8] = s[4] | (s[6] « 4);  // 4 mvs
        d[8] = d[4] + d[6];

        s[6] = (d[4] + d[2] < d[0] + d[6]) ? s[4] : (s[6] « 4);  // 3 mvs
        d[6] = (d[4] + d[2] < d[0] + d[6]) ? (d[4] + d[2]) < (d[0] + d[6]);

        s[4] = 0;  // 2 mvs
        d[4] = d[0] + d[2];

        if ( d[ 6] > d[ 4] ) d[ 6] = d[ 4];
        if ( d[ 8] > d[ 6] ) d[ 8] = d[ 6];
        if ( d[10] > d[ 8] ) d[10] = d[ 8];
        if ( d[12] > d[10] ) d[12] = d[10];
        d[14] = d[12];
        if ( d[16] > d[12] ) d[16] = d[12];

        s ++; d ++;
```

```
    }
    s = ShapeList;
    d = DistoList;
    * MinorDisto = 2048;
    for ( k=0; k<8; k++ ) {
        n = MaxNumMVs - k;
        if ( (n>=2 && n<=8) <2 ) {
            dist = d[(k « 1) + 1] + d[n « 1];
            if ( dist < *MinorDisto ) {
                *MinorDisto = dist;
                best0 = (n « 1);
                best1 = (k « 1) + 1;
            }
        }
    }
    while ( best0 > 1 && d[best0] == d[best0-2] ) best0 -= 2;
    while ( best1 > 1 && d[best1] == d[best1-2] ) best1 -= 2;
    *MinorShape = s[best0] | (s[best1] « 2);
}
```

Major Shape Decision Prior to FME

Now considering the best of each 8x8 is done, and we have the total cost-adjusted-distortion for this sub-block level partition. Now among the four choices: the resulting 8x8 sub-partitioning, one 16x16, two 16x8, and two 8x16, the one gives the best cost-adjusted-distortion, will determine the final decision of partitioning shape. Any among these four, if its cost-adjusted-distortion is within *the intermediate tolerance* (which is a predefined system state) from the best distortion will be marked as **candidate shape**s.

Notice that, when the intermediate tolerance is set to 0, only the best shape will be selected as the candidate. When the intermediate tolerance is large, all four shapes will become candidates.

Assume we have all the distortions for majors enumerated in DistoMajor[k], where k = 0, 1, 2, 3, 4, and 5, for 16x16, 16x8, 8x16, the combined minors, 16x8 field, and 8x8 field respectively. Assume BestDisto is equal to the minimal of the six values DistoMajor[k], for k = 0, …5. Assume the intermediate tolerance is IntTol, the major shape k is a candidate shape if and only if DistoMajor[k]<=BestDisto+IntTol.

Shape Update after FME

Among all the candidate shapes, we recheck the distortion, if any of them is no longer with in the intermediate tolerance **DistortionTolerance** from the best choice; we drop it for reduced calculation.

Final Code Decision after BME

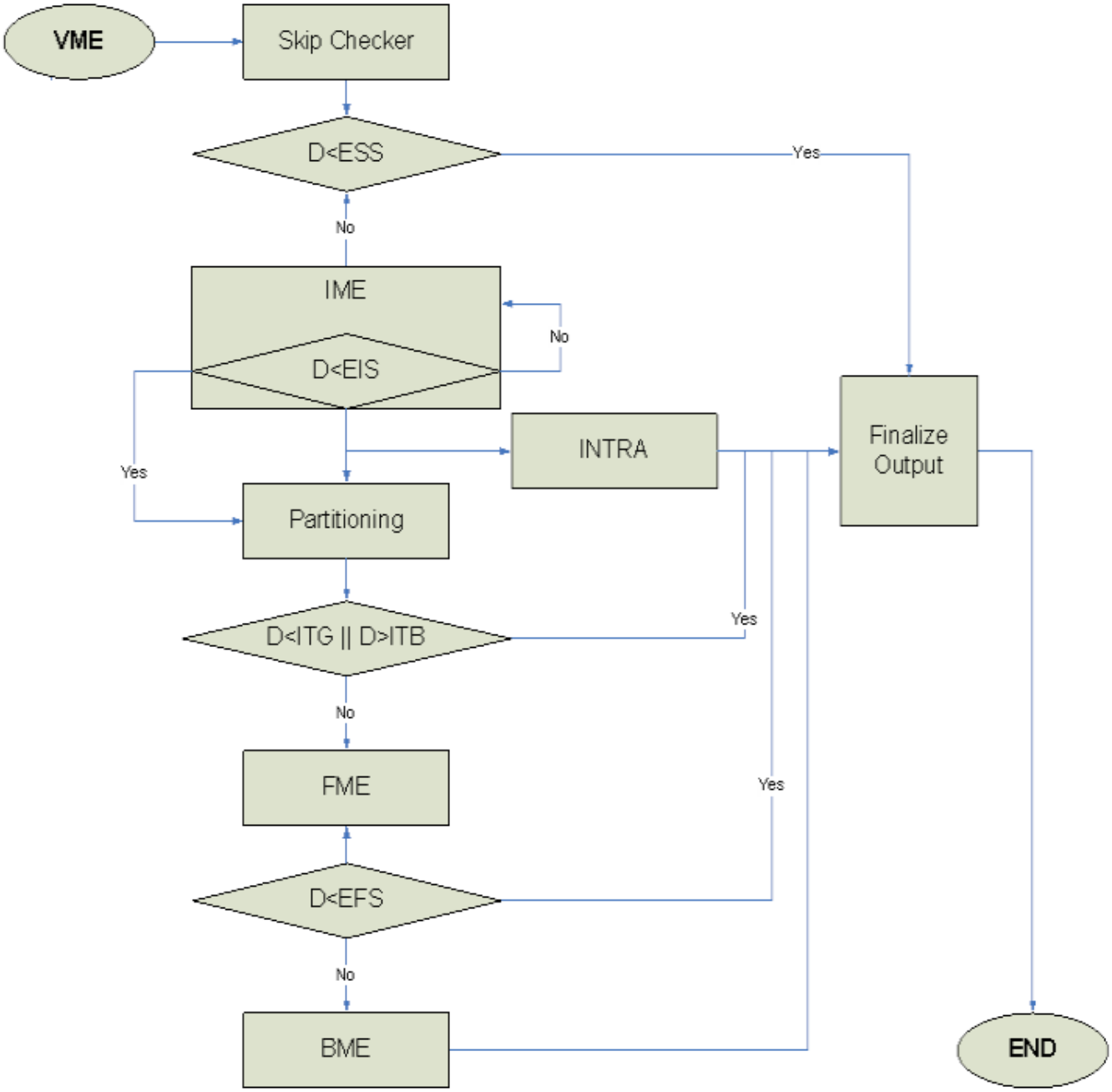
For any given candidate shape, for each motion vector, if we do have improved distortion by switch from the single direction to bi-direction, then we do it, unless the increased number of motion vectors hits above **MaxNumMVs**; in this case, we take as many as possible first the ones generate the most improvement.

Then, we choose the best among the improved candidate shapes.

Early Decisions

There are 5 programmable early decision states are available for fine control of the VME process. All stored in one byte of U4U4 format to representing a value of (B«S), (where B, called **base**, is the 4-LSB of the byte and S, called **shift**, is the 4-MSB of the byte), they are the following:

1. ESS: EarlySkipSuccess = Early successful return after Skip is checked
2. EIS: EarlyImeStop = Early IME stop when a good match is found inside of IME process.
3. ITG: ImeTooGood = Early successful return after IME is done when a good enough match is found.
4. ITB: ImeTooBad = Early termination do skip fractional and bidirectional refinement after IME is done with a hopelessly bad match as the best result.
5. EFS: EarlyFmeSuccess = Early Success after Fractional ME to skip bidirectional search.



For any reason, if all possible code types are not chosen, VME will return Intra16x16 type with all modes set to 0, and the **MinDist** is set to 0x3FFF.

Surfaces

The data elements accessed by VME are called “surfaces”. Surfaces are accessed using the surface state model.

VME uses the binding table to bind indices to surface state, using the same mechanism used by the sampling engine. A **Binding Table Index** (specified in the message descriptor) of less than 255 is used to index into the binding table, and the binding table entry contains a pointer to the SURFACE_STATE. SURFACE_STATE contains the parameters defining the surface to be accessed, including its location, format, and size.

State

This topic is currently under development.

BINDING_TABLE_STATE

VME uses the binding table to retrieve surface state. Refer to *Sampling Engine* for the definition of this state.

SURFACE_STATE

VME uses the surface state for current and reference surfaces. Refer to *Sampling Engine* for the definition of this state.

VME_STATE

This state structure contains the state used by the VME engine for data processing. VME state contains the motion search path location tables and rate-distortion weight look-up-tables. As the two sets of tables are fairly large, they are accessed as two separate states via state indexing mechanism so that applications can inter-mix the use of the search path tables and RDLUT tables.

Even though VME engine has its unique shared function ID (see Target Function ID field in the SEND instruction), the VME state is delivered through the Sampler State Pointer. When the General Purpose Pipe is used, the **Sampler State Pointer** is programmed in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command and delivered directly to Sampler/VME by hardware. This posts one usage limitation. As the VME state is overloaded on top of the Sampler State Pointer, VME messages cannot be intermixed with other Sampler messages.

Each VME state may contain up to 8 VME_SEARCH_PATH_LUT_STATE. When multiple VME_SEARCH_PATH_LUT_STATE are used, they need to be stored in memory contiguously. Each VME_SEARCH_PATH_LUT_STATE contains 32 dwords in comparison of 4 dwords of a Sampler State. When enabling sampler state pre-fetch (programming the **Sampler Count** field in the MEDIA_INTERFACE_DESCRIPTOR_LOAD command), one VME_SEARCH_PATH_LUT_STATE is equivalent to 8 Samplers. Hardware may support up to two VME_SEARCH_PATH_LUT_STATE to be pre-fetched (See See 3D_Media_GPGPU chapter, Media_GPGPU_Pipeline for more details).

VME_SEARCH_PATH_LUT_STATE

Up to eight VME_SEARCH_PATH_LUT_STATE allowed for a message to select. Each state contains one set of search path locations, and four sets of rate distortion cost function LUT for various modes and rate distortion cost function LUT for motion vectors (relative to 'cost center'). Motion vector cost

function is provided as a piece-wise-linear curve with only the values of the power-of-2 positions provided.

DWord	Bit	Description
0:13		**Search Path**
0	31:24	Search Path Location [3] (X, Y) – Relative distance from location [2]
	23:16	Search Path Location [2] (X, Y) – Relative distance from location [1]
	15:8	Search Path Location [1] (X, Y) – Relative distance from location [0]
	7:4	Search Path location [0] (Y) – specifies relative Y distance of the **next walk** from the starting position in unit of Search Unit (SU) in U4 Format = U4, (e.g. 0x3 + 0xE = 0x1)
	3:0	Search Path Distance [0] (X) – specifies relative X distance of the next walk from the starting position in unit of SU. Format = U4
1:13		Search Path Location [4 – 55] (X, Y)
14:31		**RD LUT SET 0-4**
14	31:24	LUT_MbMode [9] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [8] for Set 1 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [9] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
15	31:24	LUT_MbMode [9] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [8] for Set 3 Format = U4U4 (encoded value must fit in 12-bits)

DWord	Bit	Description
	15:8	LUT_MbMode [9] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [8] for Set 2 Format = U4U4 (encoded value must fit in 12-bits)
16	31:24	LUT_MbMode [3] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	23:16	LUT_MbMode [2] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	15:8	LUT_MbMode [1] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
	7:0	LUT_MbMode [0] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
17	31:24	LUT_MbMode [7] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MbMode [6] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MbMode [5] for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MbMode [4] for Set 0 Format = U4U4 (encoded value must fit in 12-bits)
18	31:24	LUT_MV [3] – For MV = 4 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [2] – For MV = 2 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MV [1] – For MV = 1 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)

DWord	Bit	Description
	7:0	LUT_MV [0] – For MV = 0 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
19	31:24	LUT_MV [7] – For MV = 64 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	23:16	LUT_MV [6] – For MV = 32 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	15:8	LUT_MV [5] – For MV = 16 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
	7:0	LUT_MV [4] – For MV = 8 for Set 0 Format = U4U4 (encoded value must fit in 10-bits)
20-23		**Finish RD LUT SET 1**
24-27		**Finish RD LUT SET 2**
28-31		**Finish RD LUT SET 3**

The assignment of LUT_MbMode entries is according to the MbTypeEx definition:

Index to LUT_MbMode	MbTypeEx	Description	AVC	VC1	MPEG2
0	MODE_INTRA_NONPRED	For INTRA8x8 and INTRA4x4 only. Added per 8x8 for INTRA8x8, and per 4x4 for INTRA4x4	Yes	n/a	n/a
1	MODE_INTRA MODE_INTRA_16x16	Added per 16x16 macroblock	Yes	Yes	Yes
2	MODE_INTRA_8x8	Added per 16x16 macroblock	Yes	n/a	n/a
3	MODE_INTRA_4x4	Added per 16x16 macroblock	Yes	n/a	n/a
8	MODE_INTER MODE_INTER_16x16	Added per 16x16 macroblock	Yes	Yes	Yes
9	MODE_INTER_BWD	Added for RefIdx (per partition for major type or 8x8 for minor types)	Yes	Yes	Yes

Index to LUT_MbMode	MbTypeEx	Description	AVC	VC1	MPEG2
4	MODE_INTER_16x8 MODE_INTER_8x16	Added per 16x16 macroblock	Yes	n/a	n/a
5	MODE_INTER_8x8q	Added per 8x8 subblock	Yes	Yes	n/a
6	MODE_INTER_8x4q	Added per 8x8 subblock	Yes	n/a	n/a
6	MODE_INTER_4x8q	Added per 8x8 subblock	Yes	n/a	n/a
7	MODE_INTER_4x4q	Added per 8x8 subblock	Yes	n/a	n/a
6	MODE_INTER_FIELD_16x8	Added per 16x16 macroblock	n/a	?	Yes
7	MODE_INTER_FIELD_8x8q	Added per 16x16 macroblock	n/a	n/a	n/a

The value of each byte of the LUTs will be viewed as a pair of 4-bit units: (shift, base), and constructed as

base « shift.

For example, an entry 0x4A represents the value (0xA«0x4) = 10*16 = 160. Encoded value must fit in **12**-bits (unsigned number); otherwise, the hardware behavior is undefined.

The only exception is for Index of 9, MODE_INTER_BWD, which is used as a bias for the two search directions. It is a signed number instead, in the form of (SU3U4) = (sign, shift, base). The sign bit indicates whether the bias is added to the forward (if sign = 1) or the backward (if sign = 0). The bias has a magnitude of (base « shift), which has 11-bits precision. It should be noted that the number is always added, there is no subtraction.

Intra Modes only apply to AVC standard. The mode penalty doesn't apply to Skip Mode Checking. Note that while other mode penalty applies to a fixed macroblock partition, MODE_INTRA_NONPRED applies to all three intra modes. It is a constant cost adder for intra-mode coding regardless of the block size.

For source block that is less than 16x16 (like a 16x8 source block), the proper mode penalty that is stated as "added per 16x16 macroblock" is added once to the source block (like MODE_INTER_16x8 is added once to a 16x8 source block). It will not be divided by the source block size.

The LUT_MV is added to all motion vector coordinate deltas in quarter-pel unit except for the SKIP mode, which no costing penalty applies. Given motion vector coordinate, e.g. *mvx*, which is in quarter-pel precision (S5.2), the mv delta is defined to be its difference from the given costing center, e.g. *ccx*, and the costing penalty is applied to $dx = |mvx - ccx|$. The cost penalty is a piecewise linear interpolation from the LUT_MV table whereas the values on power-of-2 integer samples are provided. The piecewise linear interpolation is performed using quarter-pel precision, while the LUT_MV are only provided for the given power-of-2 integer positions. The maximum distance provided in the table is 64 pixels. A linear ramp with gradient of 1 on integer distance is applied for bigger distances with maximum penalty capped to 0x3FF (10 bits). Thus

Costing_penalty_*x* = LUT_MV[int(*dx*)], if $dx < 3$ and $dx = \text{int}(dx)$;

Costing_penalty_*x* = LUT_MV[p+1], else if $dx = 2^p$, for any p£6;.

Costing_penalty_x = LUT_MV[p+1] + ((LUT_MV[p+2] – LUT_MV[p+1])*k)»p,
else if $dx = 2^p+k$, for any $p<6$ and $k<.2^p$, and

Costing_penalty_x = min (LUT_MV[7] + int(dx)– 64, 255), else if $dx > 64$.

The total costing penalty for a motion vector is

Costing_penalty = Costing_penalty_x + Costing_penalty_y

As a convention, a (0,0) relative search path distance (meaning a repeat search path location) is treated as the ending of the search path. Or the search path may also end when **Max Predetermined Search Path Length** is reached, or one of the Early Success conditions is reached**.**

Software must program the search path to terminate with at least one (0,0).

Software Interface – Bspec Highlights

This topic is currently under development.

Message Structure Overview

- The contents of each message are different, but they have structural similarities to reduce coding complexity.
- The first 3 input phases *('Message Phase' == 1 GRF of the message payload*) are structurally the same, given the mnemonic "Universal". Individual fields within the Universal phase are ignored based on message type.
- Additional input phases are appended to each message type to fulfill the required inputs only exclusive to that message type.
- Specifically, 4 message phases are appended to SIC (SIC0-SIC3), either 2, 4 or 6 message phases (based on streamin\streamout) are appended to IME (IME0-IME5), and 4 message phases are appended to FBR (FBR0-FBR3).
- The programmer is required to pack the necessary GRFs together to generate the correct message phase sequence before calling VME (i.e. 7 phases for SIC; 5, 7, or 9 phases for IME; and 7 phases for FBR).
- The return data is structurally common for all 3 message types, given in 7 phases. The only exception is IME return data when streamout data is present, then 2 or 4 additional phases are returned.
- Additionally, the placement of individual fields within the message phases is generally identical to that of previous generations.

IME and IDM Message Descriptor

Bits	Description
19	**Header Present.** If set, indicates that the message includes the header. This bit must be 1 for all VME messages. Format = Enable
18	Reserved: MBZ
17	**Stream-Out Enable 2nd-Best.** If set, additional message phases of record stream-out are present with the output of IME message containing the 2nd best candidates for each shape partition: 4 additional phases only when search control (M0.3 10:8) is 111b (dual reference & dual record) and 2 additional phases otherwise. Format = Enable
16	**Stream-In Enable.** If set, additional message phases of record stream-in are present with the input of IME message: 4 additional phases only when search control (M0.3 10:8) is 111b (dual reference & dual record) and 2 additional phases otherwise. Format = Enable
15	**Stream-Out Enable.** If set, additional message phases of record stream-out are present with the output of IME message: 4 additional phases only when search control (M0.3 10:8) is 111b (dual reference & dual record) and 2 additional phases otherwise. Format = Enable
14:13	**Message Type** 00: IDM [CHV, BSW] 01: Reserved 10: IME 11: Reserved
12:8	Reserved: MBZ
7:0	**Binding Table Index.** Specifies the index into the binding table for the source surface. Format = U8 Range = [0,254]

Input GRFs

GRF	Name	Msgs	New	Major Contents
0	Uni0	ALL	No	Universal control data
1	Uni1	ALL	No	Universal control data
2	Uni2	ALL	No	Costs, FT Matrix, FBR Modes
3	Uni3	ALL	Yes	8 CostCenter MVs
4	SIC0	SIC	No	8 Skip MVs
5	SIC1	SIC	No	Luma intra pix, modes, masks
6	SIC2	SIC	No	Luma intra pix, modes, masks
7	SIC3	SIC	No	Chroma intra pix & masks
8	IME0	IME	No	Search Path
9	IME1	IME	No	Search Path
10	IME2	IME	No	Streamin\Streamout
11	IME3	IME	No	Streamin\Streamout
12	IME4	IME	No	Streamin\Streamout
13	IME5	IME	No	Streamin\Streamout
14	FBR0	FBR	No	8 Inter 4x4 MVs
15	FBR1	FBR	No	8 Inter 4x4 MVs
16	FBR2	FBR	No	8 Inter 4x4 MVs
17	FBR3	FBR	No	8 Inter 4x4 MVs
18	IDM0	IDM	Yes	Source Pixel Mask

Input Message Phases by Type

VME message types require only a subset of the total GRFs of control data.

Phase	SIC	IME	FBR	IDM
0	Uni0	Uni0	Uni0	Uni0
1	Uni1	Uni1	Uni1	Uni1
2	Uni2	Uni2	Uni2	Uni2
3	Uni3	Uni3	Uni3	Uni3
4	SIC0	IME0	FBR0	IDM0
5	SIC1	IME1	FBR1	
6	SIC2	IME2	FBR2	
7	SIC3	IME3	FBR3	
8		IME4		
9		IME5		

Output GRFs

GRF	Name	Msgs	New	Major Contents
0	Ret0	ALL	No	Best MB Control Data
1	Ret1	ALL	No	8 Inter 4x4 MVs
2	Ret2	ALL	No	8 Inter 4x4 MVs
3	Ret3	ALL	No	8 Inter 4x4 MVs
4	Ret4	ALL	No	8 Inter 4x4 MVs
5	Ret5	ALL	No	Inter Block Distortions
6	Ret6	ALL	No	Block Ref Indices & FTQ Data
7	IME2	IME	No	Streamin/Streamout
8	IME3	IME	No	Streamin/Streamout
9	IME4	IME	No	Streamin/Streamout
10	IME5	IME	No	Streamin/Streamout
11	IDM0	IDM	Yes	16 Search Point Dists

12	IDM1	IDM	Yes	16 Search Point Dists
13	IDM2	IDM	Yes	16 Search Point Dists
14	IDM3	IDM	Yes	16 Search Point Dists
15	IDM4	IDM	Yes	16 Search Point Dists
16	IDM5	IDM	Yes	16 Search Point Dists
17	IDM6	IDM	Yes	16 Search Point Dists
18	IDM7	IDM	Yes	16 Search Point Dists
19	IDM8	IDM	Yes	16 Search Point Dists
20	IDM9	IDM	Yes	16 Search Point Dists
21	IDM10	IDM	Yes	16 Search Point Dists
22	IDM11	IDM	Yes	16 Search Point Dists
23	IDM12	IDM	Yes	16 Search Point Dists
24	IDM13	IDM	Yes	16 Search Point Dists
25	IDM14	IDM	Yes	16 Search Point Dists
26	IDM15	IDM	Yes	16 Search Point Dists

Output Message Phases by Type

All message types return 7 phases except IDM. IME returns 2 or 4 additional phases of streamout if it is enabled (2 for uni, 4 for bi). Note the IME streamout message phases are structurally identical to the IME streamin phases. The IDM message will return only the 16 phases of distortion mesh output.

Phase	SIC	IME	FBR	IDM
0	Ret0	Ret0	Ret0	IDM0
1	Ret1	Ret1	Ret1	IDM1
2	Ret2	Ret2	Ret2	IDM2
3	Ret3	Ret3	Ret3	IDM3
4	Ret4	Ret4	Ret4	IDM4
5	Ret5	Ret5	Ret5	IDM5

6	Ret6	Ret6	Ret6	IDM6
7		IME2		IDM7
8		IME3		IDM8
9		IME4		IDM9
10		IME5		IDM10
11				IDM11
12				IDM12
13				IDM13
14				IDM14
15				IDM15

Binding Table Pointers

The following gives the driver and HW perspective of how the RefID will map to the binding table pointers indices (and hence surface state). The fixed mapping simplifies the HW definition.

Progressive Content

	Driver View			Universal Input M1.6 RefIDs (4b Value per Block)								Conversion
BTI	Direction	Number	Field	FWD 0	BWD 0	FWD 1	BWD 1	FWD 2	BWD 2	FWD 3	BWD 3	BTI Equation
0	Source	N/A	N/A	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	*N/A*	= From input
1	FWD	0	*N/A*	0	*N/A*	0	*N/A*	0	*N/A*	0	*N/A*	= RefID * 2 + 1
2	BWD	0	*N/A*	*N/A*	0	*N/A*	0	*N/A*	0	*N/A*	0	= RefID * 2 + 2
3	FWD	1	*N/A*	1	*N/A*	1	*N/A*	1	*N/A*	1	*N/A*	= RefID * 2 + 1
4	BWD	1	*N/A*	*N/A*	1	*N/A*	1	*N/A*	1	*N/A*	1	= RefID * 2 + 2
5	FWD	2	*N/A*	2	*N/A*	2	*N/A*	2	*N/A*	2	*N/A*	= RefID * 2 + 1
6	BWD	2	*N/A*	*N/A*	2	*N/A*	2	*N/A*	2	*N/A*	2	= RefID * 2 + 2
7	FWD	3	*N/A*	3	*N/A*	3	*N/A*	3	*N/A*	3	*N/A*	= RefID * 2 +

	Driver View			Universal Input M1.6 RefIDs (4b Value per Block)								Conversion
BTI	Direction	Number	Field	FWD 0	BWD 0	FWD 1	BWD 1	FWD 2	BWD 2	FWD 3	BWD 3	BTI Equation
												1
8	BWD	3	N/A	N/A	3	N/A	3	N/A	3	N/A	3	= RefID * 2 + 2
9	FWD	4	N/A	4	N/A	4	N/A	4	N/A	4	N/A	= RefID * 2 + 1
10	BWD	4	N/A	N/A	4	N/A	4	N/A	4	N/A	4	= RefID * 2 + 2
11	FWD	5	N/A	5	N/A	5	N/A	5	N/A	5	N/A	= RefID * 2 + 1
12	BWD	5	N/A	N/A	5	N/A	5	N/A	5	N/A	5	= RefID * 2 + 2
13	FWD	6	N/A	6	N/A	6	N/A	6	N/A	6	N/A	= RefID * 2 + 1
14	BWD	6	N/A	N/A	6	N/A	6	N/A	6	N/A	6	= RefID * 2 + 2
15	FWD	7	N/A	7	N/A	7	N/A	7	N/A	7	N/A	= RefID * 2 + 1
16	BWD	7	N/A	N/A	7	N/A	7	N/A	7	N/A	7	= RefID * 2 + 2
17	FWD	8	N/A	8	N/A	8	N/A	8	N/A	8	N/A	= RefID * 2 + 1
18	BWD	8	N/A	N/A	8	N/A	8	N/A	8	N/A	8	= RefID * 2 + 2
19	FWD	9	N/A	9	N/A	9	N/A	9	N/A	9	N/A	= RefID * 2 + 1
20	BWD	9	N/A	N/A	9	N/A	9	N/A	9	N/A	9	= RefID * 2 + 2

	Driver View			Universal Input M1.6 RefIDs (4b Value per Block)								Conversion
BTI	Direction	Number	Field	FWD 0	BWD 0	FWD 1	BWD 1	FWD 2	BWD 2	FWD 3	BWD 3	BTI Equation
21	FWD	10	N/A	10	N/A	10	N/A	10	N/A	10	N/A	= RefID * 2 + 1
22	BWD	10	N/A	N/A	10	N/A	10	N/A	10	N/A	10	= RefID * 2 + 2
23	FWD	11	N/A	11	N/A	11	N/A	11	N/A	11	N/A	= RefID * 2 + 1
24	BWD	11	N/A	N/A	11	N/A	11	N/A	11	N/A	11	= RefID * 2 + 2
25	FWD	12	N/A	12	N/A	12	N/A	12	N/A	12	N/A	= RefID * 2 + 1
26	BWD	12	N/A	N/A	12	N/A	12	N/A	12	N/A	12	= RefID * 2 + 2
27	FWD	13	N/A	13	N/A	13	N/A	13	N/A	13	N/A	= RefID * 2 + 1
28	BWD	13	N/A	N/A	13	N/A	13	N/A	13	N/A	13	= RefID * 2 + 2
29	FWD	14	N/A	14	N/A	14	N/A	14	N/A	14	N/A	= RefID * 2 + 1
30	BWD	14	N/A	N/A	14	N/A	14	N/A	14	N/A	14	= RefID * 2 + 2
31	FWD	15	N/A	15	N/A	15	N/A	15	N/A	15	N/A	= RefID * 2 + 1
32	BWD	15	N/A	N/A	15	N/A	15	N/A	15	N/A	15	= RefID * 2 + 2

Interlaced Content

	Driver View			Universal Input M1.6 RefIDs (4b Value per Block)								Conversion
BTI	Direction	Number	Field	FWD 0	BWD 0	FWD 1	BWD 1	FWD 2	BWD 2	FWD 3	BWD 3	BTI Equation
0	Source	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	= From input
1	FWD	0	Top	0	N/A	0	N/A	0	N/A	0	N/A	= RefID * 2 + 1
2	BWD	0	Top	N/A	0	N/A	0	N/A	0	N/A	0	= RefID * 2 + 2
3	FWD	0	Bot	1	N/A	1	N/A	1	N/A	1	N/A	= RefID * 2 + 1
4	BWD	0	Bot	N/A	1	N/A	1	N/A	1	N/A	1	= RefID * 2 + 2
5	FWD	1	Top	2	N/A	2	N/A	2	N/A	2	N/A	= RefID * 2 + 1
6	BWD	1	Top	N/A	2	N/A	2	N/A	2	N/A	2	= RefID * 2 + 2
7	FWD	1	Bot	3	N/A	3	N/A	3	N/A	3	N/A	= RefID * 2 + 1
8	BWD	1	Bot	N/A	3	N/A	3	N/A	3	N/A	3	= RefID * 2 + 2
9	FWD	2	Top	4	N/A	4	N/A	4	N/A	4	N/A	= RefID * 2 + 1
10	BWD	2	Top	N/A	4	N/A	4	N/A	4	N/A	4	= RefID * 2 + 2
11	FWD	2	Bot	5	N/A	5	N/A	5	N/A	5	N/A	= RefID * 2 + 1
12	BWD	2	Bot	N/A	5	N/A	5	N/A	5	N/A	5	= RefID * 2 + 2
13	FWD	3	Top	6	N/A	6	N/A	6	N/A	6	N/A	= RefID * 2 + 1
14	BWD	3	Top	N/A	6	N/A	6	N/A	6	N/A	6	= RefID * 2 + 2
15	FWD	3	Bot	7	N/A	7	N/A	7	N/A	7	N/A	= RefID * 2 + 1
16	BWD	3	Bot	N/A	7	N/A	7	N/A	7	N/A	7	= RefID * 2 + 2
17	FWD	4	Top	8	N/A	8	N/A	8	N/A	8	N/A	= RefID * 2 + 1

	Driver View			Universal Input M1.6 RefIDs (4b Value per Block)								Conversion
BTI	Direction	Number	Field	FWD 0	BWD 0	FWD 1	BWD 1	FWD 2	BWD 2	FWD 3	BWD 3	BTI Equation
18	BWD	4	Top	N/A	8	N/A	8	N/A	8	N/A	8	= RefID * 2 + 2
19	FWD	4	Bot	9	N/A	9	N/A	9	N/A	9	N/A	= RefID * 2 + 1
20	BWD	4	Bot	N/A	9	N/A	9	N/A	9	N/A	9	= RefID * 2 + 2
21	FWD	5	Top	10	N/A	10	N/A	10	N/A	10	N/A	= RefID * 2 + 1
22	BWD	5	Top	N/A	10	N/A	10	N/A	10	N/A	10	= RefID * 2 + 2
23	FWD	5	Bot	11	N/A	11	N/A	11	N/A	11	N/A	= RefID * 2 + 1
24	BWD	5	Bot	N/A	11	N/A	11	N/A	11	N/A	11	= RefID * 2 + 2
25	FWD	6	Top	12	N/A	12	N/A	12	N/A	12	N/A	= RefID * 2 + 1
26	BWD	6	Top	N/A	12	N/A	12	N/A	12	N/A	12	= RefID * 2 + 2
27	FWD	6	Bot	13	N/A	13	N/A	13	N/A	13	N/A	= RefID * 2 + 1
28	BWD	6	Bot	N/A	13	N/A	13	N/A	13	N/A	13	= RefID * 2 + 2
29	FWD	7	Top	14	N/A	14	N/A	14	N/A	14	N/A	= RefID * 2 + 1
30	BWD	7	Top	N/A	14	N/A	14	N/A	14	N/A	14	= RefID * 2 + 2
31	FWD	7	Bot	15	N/A	15	N/A	15	N/A	15	N/A	= RefID * 2 + 1
32	BWD	7	Bot	N/A	15	N/A	15	N/A	15	N/A	15	= RefID * 2 + 2

RDE Packet Mapping

	Input Packets	
Phase	RDE-Intra	RDE-Inter
0	Uni0	Uni0
1	Uni1	Uni1
2	Uni2	Uni2
3	SIC0	SIC0
4	Cost0	Cost0
5	Cost1	Cost1
6	Cost2	Cost2
7	CU0	CU0
8	NP0	NMV0
9	NP1	NMV1
10	NP2	NMV2
11	NP3	NMV3
12	NP4	NMV4
13	NP5	
14	NP6	
15	NP7	
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		

	Output Packets	
Phase	RDE-B	RDE-A
0	Ret0	Ret0
1	Ret1	Ret1
2	CU0	CU0
3		RP0
4		RP1
5		RP2
6		RP3
7		RP4
8		RP5
9		RP6
10		RP7
11		
12		
13		
14		
15		
16		
17		
18		
19		
20		
21		
22		
23		
24		
25		
26		
27		
28		
29		
30		
31		

Glossary of Messages

This section describes the glossary of messages in regard to Media Sampler.

Universal Input Message Phases

Major changes from the previous generation:

- Many fields are only required for one or two of the message types.
- MV cost and mode cost are moved into the message payload.
- RefID per block are new inputs.
- Enables for forward transform skip check, chroma searching.
- Thresholds and control data for forward transform skip check.
- Many of the performance thresholds have been removed (IME success, skip success, etc).

ValidMsgType = "..." identifies the given field is required for each message type. Hardware ignores these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

DWord	Bits	Description
M0.5	31:24	Reference Region Height (RefHeight): This field specifies the reference region height in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in vertical direction. The value must be a multiple of 4. ValidMsgType = IME Format = U8 Range = [8, 64]
	23:16	Reference Region Width (RefWidth): This field specifies the search region width in pixels. When bidirectional search is enabled, this applies to both search regions. Minus 16 provides the number of search point in horizontal direction. The value must be a multiple of 4. ValidMsgType = IME Format = U8 Range = [20, 128] Note: Please make sure the reference windows are not completely outside of the video frame. In that case, VME behavior is undefined. Note: Reference Window size must be <= Surface Size, otherwise VME behavior is undefined.
	15:8	Ignored
	7:0	Dispatch ID. This ID is assigned by the fixed function unit and is a unique identifier for the thread. It is used to free up resources used by the thread upon thread completion.

DWord	Bits	Description
		ValidMsgType = SIC, IME, FBR
M0.4	31:0	Ignored (reserved for hardware delivery of binding table pointer)
M0.3	31	Reserved: MBZ
	30:24	Sub-Macroblock Sub-Partition Mask (SubMbPartMask): This field defines the bit-mask for disabling sub-partition and sub-macroblock modes. The lower 4 bits are for the major partitions (sub-macroblock) and the higher 3 bits for minor partitions (with sub-partition for 4x(8x8) sub-macroblocks. xxxxxx1 : 16x16 sub-macroblock disabled xxxxx1x : 2x(16x8) sub-macroblock within 16x16 disabled xxxx1xx : 2x(8x16) sub-macroblock within 16x16 disabled xxx1xxx : 1x(8x8) sub-partition for 4x(8x8) within 16x16 disabled xx1xxxx : 2x(8x4) sub-partition for 4x(8x8) within 16x16 disabled x1xxxxx : 2x(4x8) sub-partition for 4x(8x8) within 16x16 disabled 1xxxxxx : 4x(4x4) sub-partition for 4x(8x8) within 16x16 disabled 1111111: Invalid Note: Invalid to have all partions disabled in the IME call. ValidMsgType = IME Usage Note: One example usage of only enabling 4x(4x4) sub-partition while all other partitions are disabled is for video processing, where parallel motion searches are performed for 16 4x4 blocks. For that no further block combination (into larger sub-partitions/sub-macroblocks) is needed.
	23:22	Intra SAD Measure Adjustment (IntraSAD): This field specifies distortion measure adjustments used for the motion search SAD comparison. This field applies to both luma and chroma intra measurement. 00b: None 01b: Reserved 10b: Haar transform adjusted 11b: Reserved ValidMsgType = SIC
	21:20	Inter SAD Measure Adjustment (InterSAD):This field specifies distortion measure adjustments used for the motion search SAD comparison. This field applies to both luma and chroma intra measurement. 00b: None

DWord	Bits	Description
		01b: Reserved 10b: Haar transform adjusted 11b: Reserved ValidMsgType = SIC, IME, FBR, IDM Note: IDM msgs cannot have InterSAD set to 10b (Haar transform adjusted) if IdmSrcPixelMask is used. InterSAD must be set to 00b (None) if either IDMShapeMode5x5 or IDMShapeMode7x7 is enabled.
	19	Block-Based Skip Enabled: When this field is set on the skip thresholding passing criterion will be based on the maximal distortion of individual blocks (8x8's or 4x4's) instead of their sum (i.e. the distortion of 16x16). The block size is 8x8 if and only if the Transform 8x8 Flag is set to ON and the source size is 16x16. ValidMsgType = SIC
	18	BME disable for FBR Message (BMEDisableFBR): FBR messages that do not want bidirectional motion estimation performed will set this bit and VME will only perform fractional refinement on the shapes identified by subpredmode. Note: only the LSB of the subpredmode for each shape will be considred in FBR (a shape is either FWD or BWD as input of FBR, output however could change to BI if BME is enabled). 0 = BME enabled 1 = BME disabled ValidMsgType = FBR
	17	Forward Transform Skip Check Enable (FTEnable): This field enables the forward transform calculation for skip check. It does not override the other skip calculations but it does decrease the performance marginially so don't enable it unless the transform is necessary. 0 = FT disabled 1 = FT enabled ValidMsgType = SIC
	16	Process Inter Chroma Pixels Mode (InterChroaZmaMode): This bit switches the inter operations from luma mode to chroma mode. All shapes sizes are referred to as UV pairs. For instance, the 4x4 shape is a 8x4 of pixel components (16 U and 16 V, interleaved vertically) and the 8x8 shape is a 16x8 of pixel components. MBMode is always 8x8. MBSubShape is either 8x8 or 4x4 indicated by LSB[1:0]. Bits[7:2] are MBZ.

DWord	Bits	Description
		For MBSubShape of 4x4, SubPredMode is mapped to each 4x4 shape. Only 8x8 and 4x4 ModeCost are valid. Source block size is ignored. Streamin/streamout distortions are overloaded on 16x16 (Chroma8x8) and 8x8 (Chroma4x4). BilinearEnable is ignored (Chroma can only perform bilinear filtering) Restrictions when set: Intra operations are disabled (SIC), valid ref window sizes are 32x20, 24x24 (max Xsus), 16x32 (max Xsus), and 10x20 (max Xsus) (IME), adaptive is disabled (IME), no backward penalty cost (ALL), and only 4x4 and 8x8 shapes are valid (ALL). ValidMsgType = SIC, IME, FBR
	15	Disable Field Cache Allocation: This field, when set to 1, disables the optimized field cache line method in the Sampler Cache for reference block data when RefAccess is 1 (field based). It is ignored by hardware if RefAccess is 0. 0 – Frame or field cache lines according to RefAccess 1 – Always frame cache lines ValidMsgType = IME, IDM
	14	**Skip Mode Type** For B_DIRECT_16x16, both motion vectors of the skip center pair 0 are used. For B_DIRECT_8x8s, all four skip center pairs are fully used (VME never tries to combine them with non-skip shapes from IME, FME, or BME). 0 : SKIP_1MVP – one MV pair for 16x16 1 : SKIP_4MVP – Four MV pairs for 8x8s (in this case and only this case, SkipCenter Delta 1-3 is used) Note: SkipModeType should be programmed to 1MVP for non-16x16 Source size. ValidMsgType = SIC
	13:12	**Sub-Pel Mode (SubPelMode)** This field defines the half/quarter pel modes. The mode is inclusive, ie., higher precision mode samples lower precision locations. 00b: Integer mode searching 01b: Half-pel mode searching 10b: Reserved 11b: Quarter-pel mode searching ValidMsgType = FBR

DWord	Bits	Description
	11	**Dual Search Path Option** Used only for dual record cases, this field flags whether two searching records uses the same or the different paths. 0: Use the same path as specified by the Search Path Location array 1: Use the different paths, the first one uses the even entries of the Search Path Location array and the second one uses the odd entries of the Search Path Location array. ValidMsgType = IME
	10:8	**Search Control (SearchCtrl)** This field specifies how the motion search is performed. ValidMsgType = IME The following table shows the valid encodings. Other encodings are reserved.

Code	Mode
000b	Single reference, single record and single start. Search is performed only on reference 0; only cost center 0 and start 0 are used. There is only one record. Adaptive search is also allowed. However, when AdaptiveEn is on, LenSU must be at least 2 as the adaptive search in VME is one-step delayed. This is the common single directional motion search mode.
001b	Single reference, single record and dual start. Search is performed only on reference 0; only cost center 0 is used. There is only one record. Search performs first on start 0 and then on start 1. Then if LenSP is not reached, the predetermined search path will start on start 1 with increment added to start 1 location. It then is followed by adaptive search. This is used for single direction adaptive search.
011b	Single reference, dual record (and implied dual start). Search is performed only on reference 0; both cost center 0 and 1 and start 0 and 1 are used. Two records are used for both paths during IME. When integer search is complete, the two records are combined to find the best search. Sub-pel refinement is only performed from the best one. This may be used for search for multiple motion search candidates/predicators.
111b	Dual reference, dual record (and implied dual start). Search is performed on references 0/1 with both cost centers 0/1 and starts 0/1. Two records are used for both paths during IME. When integer search is complete, and then sub-pel refinement is also performed

DWord	Bits	Description
		separately, the two records are combined to find the best search on a subblock basis. This may be used for bidirectional motion search, or multi-reference P search. Whether bidirectional is enabled or not depends on the bidirection sub-macroblock mask. If BiSubMbPartMask is set to 1111'b, bidirectional search is disabled. VME outputs only the best unidirectional search results. Otherwise, BME is performed. Note that bidirectional search and sub-pel refinement are orthogonal features that can be enabled independently.
	7	**Reference Access (RefAccess)** This field defines how the reference blocks are accessed from the reference frames. It indicates if the source picture is a frame picture or a field picture. Programming Note: For all known video coding standards, reference pictures always have the same picture type as the source picture. Therefore, this field should be programmed to be the same as SrcAccess. 0: Frame based 1: Field based ValidMsgType = SIC, IME, FBR, IDM
	6	**Source Access (SrcAccess)** This field defines how the source block is accessed from the source frame. It indicates if the source picture is a frame picture or a field picture. It is similar to the Picture Type used in video standards. 0: Frame based 1: Field based ValidMsgType = SIC, IME, FBR, IDM
	5:4	Inter MbType Remap (MbTypeRemap): This field controls the mapping of the output MbType when the VME output is an Inter (IntraMbFlag = INTER). The intended usage, for example, is for two forward (or backward) references or for two search regions from the same reference picture in one VME call. Hardware ignores this field if the VME output is an intra type (IntraMbFlag = INTRA). 00b: No remapping 01b: Remapping MbType to forward only (1-3 mapped to 1, even numbers in [4-14h] mapped to 4, odd numbers in [5-15h] mapped to 5, and 16h is unchanged) 10b: Remapping MbType to backward only (1-3 mapped to 2, even numbers in [4-14h] mapped to 6, odd numbers in [5-15h] mapped to 7, and 16h is unchanged) 11b: Reserved ValidMsgType = IME, FBR
	3	Reserved: MBZ

DWord	Bits	Description
	2	Reserved: MBZ
	1:0	**Source Block Size (SrcSize)** This field defines how the 16x16 source block is formed. When Source Block Size is less than 16x16, SU larger than 4x4 is used. 00b: 16x16 01b: 16x8 10b: Reserved (for 8x16) 11b: 8x8 ValidMsgType = SIC, IME, FBR, IDM Note: For IDM message, the source block size should be always programmed to 16x16.
M0.2	31:16	**Source Y (SrcY)** This field defines the vertical position (of the block's upper-left pixel) in units of pixels for the source block in the source frame. **Restriction** The Y address restriction is removed. Exception: for SIC messages where Intra Compute Type is set to 00 (Luma + Chroma enabled), SrcY must be a multiple of 2. ValidMsgType = SIC, IME, FBR, IDM Format = U16
	15:0	**Source X (SrcX)** This field defines the horizontal position (of the block's upper-left pixel) in units of pixels for the source block in the source picture. The source block must be within the source picture starting at any integer grid. For SIC messages where Intra Compute Type is set to 00 (Luma + Chroma enabled), SrcX must be a multiple of 2. ValidMsgType = SIC, IME, FBR, IDM Format = U16
M0.1	31:16	**Reference 1 Y Delta (Ref1Y)** This field defines the vertical position (of the upper-left corner of the reference region) in units of pixels for the Reference 1 region relative to the surface origin.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>
<table>
<tr><th>Project</th><th>Restriction</th></tr>
<tr><td>CHV, BSW</td><td>The Y address restriction is removed.
ValidMsgType = IME
Format = S15
Hardware Range: [-2048 to 2047]</td></tr>
</table>
Format = U16
</td></tr>
<tr><td></td><td>15:0</td><td>
Reference 1 X Delta (Ref1X)
This field defines the horizontal position (of the upper-left corner of the reference region) in units of pixels for the Reference 1 region relative to the surface origin.
The resulting reference region is allowed to be outside the picture. Pixel replication is applied to generate out of bound reference pixels.
This field is only valid when dual reference mode is selected.
Note: For search control=3, this must equal Ref0X.
ValidMsgType = IME
Format = S15
Hardware Range: [-2048 to 2047]
</td></tr>
<tr><td>M0.0</td><td>31:16</td><td>
Reference 0 Y Delta (Ref0Y)
This field defines the vertical position (of the upper-left corner of the reference region) in units of pixels for the Reference 0 region relative to the surface origin.
<table>
<tr><th>Project</th><th>Restriction</th></tr>
<tr><td>CHV, BSW</td><td>The Y address restriction is removed.
ValidMsgType = IME, IDM
Format = S15
Hardware Range: [-2048 to 2047]</td></tr>
</table>
Format = U16
</td></tr>
<tr><td></td><td>15:0</td><td>
Reference 0 X Delta (Ref0X)
This field defines the horizontal position (of the upper-left corner of the reference region) in units of pixels for Reference 0 region relative to the surface origin.
The resulting reference region is allowed to be outside the picture. Pixel replication is applied to
</td></tr>
</table>

DWord	Bits	Description
		generate out of bound reference pixels. ValidMsgType = IME, IDM Format = S15 Hardware Range: [-2048 to 2047]
M1.7	31:24	**Skip Center Enable Mask (SkipCenterMask):** [bits 31...24] xxxx xxx1: Ref0 Skip Center 0 is enabled [corresponds to M2.0] xxxx xx1x: Ref1 Skip Center 0 is enabled [corresponds to M2.1] xxxx x1xx: Ref0 Skip Center 1 is enabled [corresponds to M2.2] xxxx 1xxx: Ref1 Skip Center 1 is enabled [corresponds to M2.3] xxx1 xxxx: Ref0 Skip Center 2 is enabled [corresponds to M2.4] xx1x xxxx: Ref1 Skip Center 2 is enabled [corresponds to M2.5] x1xx xxxx: Ref0 Skip Center 3 is enabled [corresponds to M2.6] 1xxx xxxx: Ref1 Skip Center 3 is enabled [corresponds to M2.7] Illegal cases: Disable both Ref0 and Ref1 Skip Center 0 in case of Skip_1MVP. Disable both Ref0 and Ref1 for any Skip Center pair in case of Skip_4MVP. ValidMsgType = SIC ValidMsgType = SIC
	23	IDM Shape Mode Select (IDMShapeMode): [Also see M1.1 bits 30 and 31] This bit selects what shape size the IDM is searching for. 0: 16x16 1: 8x8 ValidMsgType = IDM Note: Only ref window size of 32x32 (shape16x16), 24x24(shape8x8), 128x16 (shape16x16), and 32x16 (shape16x16) are supported. Search control[2:0] defaults to single ref and single start. Luma only.
	22	**RefID Cost Mode Select (RefIDCostMode)** Selects the RefID costing mode. 0 = Mode0 (AVC) 1 = Mode1 (linear)

DWord	Bits	Description
		ValidMsgType = SIC, IME, FBR, IDM
	21	**Enable AC-Only HAAR (AConlyHAAR)** This bit zeros out the DC component in the HAAR SATD block. 0 = AC+DC HAAR 1 = AC HAAR ValidMsgType = SIC, IME, FBR
	20	**Enable Weighted-SAD\HAAR (WeightedSADHAAR)** Project: CHV, BSW Restriction: This bit enables weighted SAD\HAAR. 0: No weighted-SAD 1: Enable weighted-SAD Note: if this bit is 1, ShapeMask is ignored and only 16x16 shapes are accumulated (no partitioning). Restrictions: Only supported for source-type luma 16x16. Only support unidirectional search (on Ref0). See M1.3 for individual sub-block weight control. ValidMsgType = IME, IDM
	19	**Source Field Polarity Select (SrcFieldPolarity)** If SrcAccess = 1 (M0.3-6), meaning field based, than the hardware requires this value to derive the correct location on the source surface in memory to fetch pixels. This is because the source is stored as a frame picture with both fields interleaved in memory and the SrcY value (M0.2-31:16) is the location on the field picture (in other words, it does not convey the field polarity). Hence, the starting y-pixel coordinate fetched from memory is: SrcY* 2 + SrcFieldPolarity Else, this field is ignored by the hardware. ValidMsgType = SIC, IME, FBR, IDM Format = U1
	18	**Bilinear Filter Enable (BilinearEnable)** If set, the fractional filter implements a simple bilinear interpolation filter instead of the 4-tap filter. Note: This is supported for both hpel and qpel interpolation.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>ValidMsgType = SIC, FBR

Format = Enable</td></tr>
<tr><td></td><td>17:16</td><td>MV Cost Scaling Factor (MVCostScaleFactor)

This term allows the user to redefine the precision of the lookup into the LUT_MV based on the MV cost difference from the cost center. The piecewise linear cost function is defined from 0 to 64 in powers of 2 intervals, and the precision of the difference is set by this field. There are 4 precision choices:

00b: Qpel [Qpel difference between MV and cost center: eff cost range 0-15pel]

01b: Hpel [Hpel difference between MV and cost center: eff cost range 0-31pel]

10b: Pel [Pel difference between MV and cost center: eff cost range 0-63pel]

11b: 2pel [2Pel difference between MV and cost center: eff cost range 0-127pel]

ValidMsgType = SIC, IME, FBR, IDM

Format = U2</td></tr>
<tr><td></td><td>15:8</td><td>Macroblock Intra Structure (MbIntraStruct): This is a bitmask that specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search.

Note: The user must set Bit6=Bit5.

The bit positions in the following table are relative positions within the field. For example, bit 7 in the table is bit 15 in the containing DWord.

<table><tr><th>Bit</th><th>MotionVerticalFieldSelect Index</th></tr><tr><td>7</td><td>Reserved: MBZ (for IntraPredAvailFlagF – F (pixel[-1,7] available for MbAff)</td></tr><tr><td>6</td><td>Reserved: MBZ (for IntraPredAvailFlagA/E – A (left neighbor top half for MbAff)</td></tr><tr><td>5</td><td>IntraPredAvailFlagE/A – A (Left neighbor or Left bottom half)</td></tr><tr><td>4</td><td>IntraPredAvailFlagB – B (Upper neighbor)</td></tr><tr><td>3</td><td>IntraPredAvailFlagC – C (Upper left neighbor)</td></tr><tr><td>2</td><td>IntraPredAvailFlagD – D (Upper right neighbor)</td></tr><tr><td>1:0</td><td>Reserved: MBZ (ChromaIntraPredMode)</td></tr></table>
ValidMsgType = SIC</td></tr>
<tr><td></td><td>7</td><td>Luma Intra Source Corner Swap (IntraCornerSwap): This field specifies the format of the intra luma neighbor pixel format in the message.

0: Top neighbors are in sequential order.

1: Left-top corner is swapped with the last left-edge neighbor.</td></tr>
</table>

DWord	Bits	Description
		ValidMsgType = SIC
	6	**Non Skip MB Mode Cost Added (NonSkipModeAdded)** This field indicates that the distortion of the survived motion vectors becomes non-skip, and the MB mode cost is added to its distortion. ValidMsgType = SIC
	5	**Non Skip Zero MV Cost Added (NonSkipZMvAdded)** This field indicates that the distortion of the survived motion vectors becomes non-skip, and the zero MV component costs are added to its distortion. ValidMsgType = SIC
	4:0	**Luma Intra Partition Mask (IntraPartMask)** This field specifies which Luma Intra partition is enabled/disabled for intra mode decision. xxxx1: luma_intra_16x16 disabled xxx1x: luma_intra_8x8 disabled xx1xx: luma_intra_4x4 disabled Note: For SIC message with IntraComputeType == 00 or 01, at least 1 partition must be enabled. Bits [4:3] MBZ ValidMsgType = SIC
M1.6	31:28	**Bwd Block 3 RefID**
	27:24	**Fwd Block 3 RefID**
	23:20	**Bwd Block 2 RefID**
	19:16	**Fwd Block 2 RefID**
	15:12	**Bwd Block 1 RefID**
	11:8	**Fwd Block 1 RefID**
	7:4	**Bwd Block 0 RefID**
	3:0	**Fwd Block 0 RefID** M1.6 contains 8 input RefIDs, 1 per block. The RefID is used to penalize selection of shapes away from the optimal RefID similar to how MVCost penalizes shapes with motion vectors far from the cost center.

Project	Restriction
CHV, BSW	Note: All 4 Bwd RefID are ignored by HW for IDM message type. Format = U4 ValidMsgType = SIC, IME, FBR, IDM

DWord	Bits	Description
	31:16	Reserved: MBZ
	15:0	Reserved: MBZ
	31:16	Reserved: MBZ
	15:0	Reserved: MBZ
M1.3	31:30	**Weighted SAD Control Sub-block 15 (F)**
	29:28	**Weighted SAD Control Sub-block 14 (E)**
	27:26	**Weighted SAD Control Sub-block 13 (D)**
	25:24	**Weighted SAD Control Sub-block 12 (C)**
	23:22	**Weighted SAD Control Sub-block 11 (B)**
	21:20	**Weighted SAD Control Sub-block 10 (A)**
	19:18	**Weighted SAD Control Sub-block 9**
	17:16	**Weighted SAD Control Sub-block 8**
	15:14	**Weighted SAD Control Sub-block 7**
	13:12	**Weighted SAD Control Sub-block 6**
	11:10	**Weighted SAD Control Sub-block 5**
	9:8	**Weighted SAD Control Sub-block 4**
	7:6	**Weighted SAD Control Sub-block 3**
	5:4	**Weighted SAD Control Sub-block 2**
	3:2	**Weighted SAD Control Sub-block 1**
	1:0	**Weighted SAD Control Sub-block 0** When the Weighted SAD control is enabled (M1.7 bit 20) these values are used to decrease the magnitude of each sub-block by dividing the 4x4 SAD\HAAR output mapped to that 4x4 of the source MB. The control value divides the 4x4 SAD\HAAR output by 2^control value. 0: »0 (div by 1) 1: »1 (div by 2) 2: »2 (div by 4) 3: »3 (div by 8) The output produces 4 16x16 macroblock results each with different weighted-SAD control. The values from M1.3 31:0 are mapped onto the sub-blocks of the source MB in the traditional Z-order for the first 16x16 weighted-SAD result: 16x16_0 Weighted-SAD Control Mapping: 0145 2367 89CD ABEF The HW will horizontally, vertically, and diagonallay map these weights from M1.3 31:0 to produce

DWord	Bits	Description
		the weights for the other 3 16x16 macroblock results. 16x16_1 Weighted-SAD Control Mapping (Horizontal reflection): 5410 7632 DC98 FEBA 16x16_2 Weighted-SAD Control Mapping (Vertical reflection): ABEF 89CD 2367 0145 16x16_3 Weighted-SAD Control Mapping (Diagonal reflection: FEBA DC98 7632 5410 ValidMsgType = IME, IDM
M1.2	31:28	**Start Center 1 Y (Start1Y)** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in units of SU. ValidMsgType = IME Format = U4
	27:24	**Start Center 1 (Start1X)** This field defines the X position of Search Path 1 relative to the reference X location. It is in units of SU. The corresponding reference block must be fully within the reference region. ValidMsgType = IME Format = U4
	23:20	**Start Center 0 Y (Start0Y)** This field defines the Y position of Search Path 1 relative to the reference Y location. It is in units of SU. ValidMsgType = IME Format = U4
	19:16	**Start Center 0 X (Start0X)**

DWord	Bits	Description
		This field defines the X position of Search Path 1 relative to the reference X location. It is in units of SU. The corresponding reference block must be fully within the reference region. ValidMsgType = IME Format = U4
	15:8	**Maximum Search Path Length (MaxNumSU)** This field defines the maximum number of SUs per reference including the predetermined SUs and the adaptively generated SUs. Note: Every SU in fixed path is counted (including the out-bound ones and repeated ones), and in addition for adaptive SUs only the ones actually searched are added. ValidMsgType = IME Format = U8, with valid range of [1,63]
	7:0	**Max Fixed Search Path Length (LenSP)** This field defines the maximum number of SUs per reference which are evaluated by the predetermined SUs. When adaptive walk is enabled, adaptive walk starts when this number is reached. Note: Every SU in fixed path is counted (including the out-bound ones and repeated ones). ValidMsgType = IME Format = U8, with valid range of [1,63]
M1.1	31	Reserved: MBZ
	30	Reserved: MBZ
	29	Ref pixel bias enable If set perform following to reference pixel: »1+64 ValidMsg type - IME, SIC, FBR, IDM
	28	Unidirectional Mix Disable (UniMixDisable): If it is on, all unidirectional resulting motion vectors must share the same direction, i.e. either all are forward, or all are backward. If this field is off, each partition, down to 8x8 subblock, may have a different mix of forward and backward motion vectors. (Within each 8x8 subblock, only one common choice is allowed.) Programming Note: For the case when BMEdisableFBR is set, only the input subpredmode direction is refined. If BMEdisableFBR is not set, both directions undergo fractional refinement before bidirectional refinement, but the subpredmode output never inverts directions if the refinement yields a better result (subpredmode could change to bidirectional in this case though). This field is MBZ except for cases of Search Control = 111'b (e.g. 7, dual reference).

DWord	Bits	Description
		ValidMsgType = IME
	27:24	Reserved: MBZ
	23:22	Reserved: MBZ [Fixed7x7Weights -- GxMask, GyMask]
	21:16	**Bidirectional Weight (BiWeight)** This field defines the weighting for the backward and forward terms to generate the bidirectional term. This field is only valid for bidirectional search (SearchCtrl = 111). ValidMsgType = SIC, FBR Format = U6 Valid Values: [16, 21, 32, 43, 48]
	15:8	**RefId Polarity Bits** Bit15->bwd block3 Bit14->bwd block2 Bit13->bwd block1 Bit12->bwd block0 Bit11->fwd block3 Bit10->fwd block2 Bit9->fwd block1 Bit8->fwd block0 ValidMsg type - IME, SIC, FBR, IDM
	7	Reserved: MBZ
	6	Extended MV Cost Range This bit specifies if the increased 12-bit mvcost range is used vs. the legacy 10-bit range. 0 = Disable 1 = Enable ValidMsgType = SIC,IME, FBR
	5:0	**Maximum Number of Motion Vectors (MaxNumMVs)** This field specifies the maximum number of motion vectors allowed for the current macroblock. This field affects the macroblock partition decision. VME returns the best partition with MvQuantity not exceeding MaxNumMVs. MaxNumMVs = 0 only allows skip as a valid Inter mode. Note: This value is used ONLY for 16x16 source MB mode.

DWord	Bits	Description
		Usage Example: Certain profiles/levels for AVC standard have restrictions for the maximum number of motion vectors allowed for two consecutive macroblocks (MaxMvsPer2Mb may be 16 or 32). ValidMsgType = IME Format = U6
M1.0	31:24	**Early IME Successful Stop Threshold (EarlyImeStop)** This field specifies the threshold value for the IME distortion computes of single 16x16 mode below which no more search is performed within the IME unit. This field only takes effect if EarlyImeSuccessEn is set. Note: Early IME exit only looks at ref0, and uses 8x8 for source 8x8 and 2 16x8 0 for source 16x8. ValidMsgType = IME Format = U4U4 (encoded value should fit in 14 bits)
	23:16	Reserved: MBZ
	15:8	Reserved: MBZ
	7	**Transform 8x8 Flag For Inter Enable (T8x8FlagForInterEn)** This field specifies whether Transform8x8Flag is updated for inter mode according to the resulting inter-mode sub-partition size. 0: Disable 1: Enable ValidMsgType = SIC, IME, FBR
	6	**X only search** This field enables searching in only the x dimension. ValidMsg type - IME, IDM
	5	**Early IME Success Enable (EarlyImeSuccessEn)** This field specifies whether the Early Success may terminate on full-pel precision. When this field is not set, if early out does occur on full-pel location, hardware continues to local sub-pel refinement search and so on. When this field is set, however, the local sub-pel refinement step is skipped and intra search is also skipped. This field only takes effect if EarlySuccessEn is set. Usage Example: This may be used for cases with large static area where (0,0) motion vector delivers very good results that no FME refinement is needed and also intra check is also skipped. This may also be used in place of Skip Mode Checking when the skip center(s) happens to be an integer location inside the SU of the Start Center(s).

DWord	Bits	Description
		0: Disable 1: Enable ValidMsgType = IME
	4:3	Reserved: MBZ
	2	Bidirectional Mix Disable (BiMixDis): If it is on, all resulting motion vectors must share the same direction, i.e. either all are unidirectional (i.e. forward or backward), or all bidirectional. If this field is off, each partition may have different search direction (forward, backward, or bidirectional). Usage Example: MPEG2 bidirectional decision is at whole macroblock level, while AVC decision is at subblock level. 0: Bidirectional decision on subblock level that bidirectional mode is enabled. 1: Bidirectional decision on whole macroblock. Note: This must be disabled for SubMbShape with any minors (8x4/4x8/4x4) in the MB. ValidMsgType = FBR
	1	Adaptive Search Enable (AdaptiveEn): This field defines whether adaptive searching is enabled for IME. When Adaptive Search is enabled, there must be at least two search steps preceded. It is either from a single start with step of >=2 or from a dual-start. 0: Disable 1: Enable ValidMsgType = IME
	0	Skip Mode Enable (SkipModeEn): This field specifies whether the skip mode checking is performed before the motion search. If this field is set, Skip Center, which may have a sub-pel precision, is first tested before IME. 0: Disable 1: Enable
M2.7	31:24	**SIC Forward Transform Coeff Threshold Matrix[6] - Highest Freq**
	23:16	**SIC Forward Transform Coeff Threshold Matrix[5]**
	15:8	**SIC Forward Transform Coeff Threshold Matrix[4]**
	7:0	**SIC Forward Transform Coeff Threshold Matrix[3]**
M2.6	31:24	**SIC Forward Transform Coeff Threshold Matrix[2]**
	23:16	**SIC Forward Transform Coeff Threshold Matrix[1]**
	15:0	**SIC Forward Transform Coeff Threshold Matrix[0]** Values of the threshold matrix[0..6] are provided here. Matrix[0] contains the DC threashold for the Forward Transform Skip check. It has increased precision vs. the other thresholds due to the larger size of DC coeffieicients. Matrix[1] through

DWord	Bits	Description
		Matrix[6] have lower precision. Threshold Matrix for 4x4 transform is as follows: 0 1 2 3 1 2 3 4 2 3 4 5 3 4 5 6 Matrix[0] Format = U16 Matrix[1..6] Format = U8 ValidMsgType = SIC
M2.5	31:24	Reserved: MBZ
	23:16	**FBR SubPredMode Input** VME uses this to select the appropriate shapes from the input message to perform FME on. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3] 00: Forward 01: Backward 10: Bidirectional 11: Illegal Note: Only the LSB of the subpredmode for each shape is considered in FBR (a shape is either FWD or BWD as input of FBR). ValidMsgType = FBR
	15:8	**FBR SubMBShape Input** This field is used to specify the subshape per block for fractional and bidirectional refinement. Bits [1:0]: SubMbShape[0] Bits [3:2]: SubMbShape[1] Bits [5:4]: SubMbShape[2] Bits [7:6]: SubMbShape[3] 00: 8x8 01: 8x4

DWord	Bits	Description
		10: 4x8 11: 4x4 ValidMsgType = FBR
	7:2	Reserved: MBZ
	1:0	**FBR MbMode Input** This field is used to specify the inter macroblock type in the same format as VME output. 00: 16x16 01: 16x8 10: 8x16 11: 8x8 ValidMsgType = FBR
M2.4	31:24	**MV 7 Cost**
	23:16	**MV 6 Cost**
	15:8	**MV 5 Cost**
	7:0	**MV 4 Cost**
M2.3	31:24	**MV 3 Cost**
	23:16	**MV 2 Cost**
	15:8	**MV 1 Cost**
	7:0	**MV 0 Cost** Motion vector costings. See 6.3.3.1 for details. In short, the cost is linearly interpolated between control points. Format = U4U4 (encoded value must fit in 10 bits) **Project** / **Note** CHV, BSW: ValidMsgType = SIC, IME, FBR, IDM
M2.2	31:24	**Chroma Intra Mode Cost** Penalty for chroma intra modes. DC = 0x Horz = 1x Vert = 1x Plane = 2x Format = U4U4 (encoded value must fit in 12-bits) ValidMsgType = SIC, IME, FBR

DWord	Bits	Description
	23:16	**RefID Cost** RefID costing base penalty. Under AVC or Linear mode, different scaling are applied on top of this. Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR, IDM
	15:8	**Mode 9 Cost** MODE_INTER_BWD Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	7:0	**Mode 8 Cost** MODE_INTER_16x16 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR, IDM
M2.1	31:24	**Mode 7 Cost** MODE_INTER_4x4q MODE_INTER_FIELD_8x8q Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
	23:16	**Mode 6 Cost** MODE_INTER_8x4q MODE_INTER_4x8q MODE_INTER_FIELD_16x8 Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
	15:8	**Mode 5 Cost** MODE_INTER_8x8q Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR, IDM
	7:0	**Mode 4 Cost** MODE_INTER_16x8

DWord	Bits	Description
		MODE_INTER_8x16 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
M2.0	31:24	**Mode 3 Cost** MODE_INTRA_4x4 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	23:16	**Mode 2 Cost** MODE_INTRA_8x8 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	15:8	**Mode 1 Cost** MODE_INTRA_16x16 Format = U4U4 (encoded value must fit in 12 bits) ValidMsgType = SIC, IME, FBR
	7:0	**Mode 0 Cost** MODE_INTRA_NONPRED Format = U4U4 (encoded value must fit in 10 bits) ValidMsgType = SIC, IME, FBR
M3.7	31:0	**BWD Cost Center 3**
M3.6	31:0	**FWD Cost Center 3**
M3.5	31:0	**BWD Cost Center 2**
M3.4	31:0	**FWD Cost Center 2**
M3.3	31:0	**BWD Cost Center 1**
M3.2	31:0	**FWD Cost Center 1**
M3.1	31:16	**BWD Cost Center 0 Delta Y (BWDCostCenter0Y)** This field defines the Y value for the first cost center relative to the picture source MB Y value for the BWD direction. All 4 Bwd Cost Center Deltas are ignored by HW for IDM message type. ValidMsgType = SIC, IME, FBR

DWord	Bits	Description
		Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	**BWD Cost Center 0 Delta X (BWDCostCenter0X)** This field defines the X value for the first cost center relative to the picture source MB X value for the BWD direction. Major shape mapping to each cost center: CC0: 16x16_0, 16x8_0, 8x16_0, 8x8_0 CC1: 8x16_1, 8x8_1 CC2: 16x8_1, 8x8_2 CC3: 8x8_3 All 4 Bwd Cost Center Deltas are ignored by HW for IDM message type. ValidMsgType = SIC, IME, FBR Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
M3.0	31:16	FWDCostCenter 0 Delta Y (FWDCostCenter0Y): This field defines the Y value for the first cost center relative to the picture source MB Y value for the FWD direction. ValidMsgType = SIC, IME, FBR, IDM Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75]
	15:0	FWDCostCenter 0 Delta X (FWDCostCenter0X): This field defines the X value for the first cost center relative to the picture source MB X value for the FWD direction. Major shape mapping to each cost center: CC0: 16x16_0, 16x8_0, 8x16_0, 8x8_0 CC1: 8x16_1, 8x8_1 CC2: 16x8_1, 8x8_2 CC3: 8x8_3 ValidMsgType = SIC, IME, FBR, IDM Format = S13.2 (2's comp)

DWord	Bits	Description
		Hardware Range: [-2048.00 to 2047.75]

SIC Input Message Phases

Major changes

- Addition of chroma pixel pairs (CbCr as 16b value) for the left 8, top 8, and top-left 1 corner.
- Addition of chroma mode masks (only 4 modes possible, so 4b mask).
- Addition of intra compute type (Y+CbCr, Y only, disabled).

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

Project	Value	Any Description
CHV, BSW	4	CHV, BSW added 1 additional universal message phase.

DWord	Bits	Name
WX+0.7	31:0	**Ref1 SkipCenter 3 Delta XY** Ref1 Skip Center 3 Delta Y: This field defines the Y value for the forward skip center relative to the 8x8 block offset from the source MB Y location in quarter-pel precision associated with Ref1. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input Y value. For SkipCenter 1: VME will add 0 to the user-input Y value. For SkipCenter 2: VME will add 32 to the user-input Y value. For SkipCenter 3: VME will add 32 to the user-input Y value. ValidMsgType = SIC Format = S13.2 (2's comp) Hardware Range: [-512.00 to 511.75] For chroma skip: Format = S12.3 (2's comp) Hardware Range: [-256.000 to 255.875] Ref1SkipCenter3 Delta X: This field defines the X value for the forward skip center relative to the 8x8 block offset from the

DWord	Bits	Name
		source MB X location in quarter-pel precision associated with Ref1. To match the relative 8x8 block location, the HW will add fixed offsets to the 4 skip centers in each direction to generate the correct pixel location to fetch the data. For SkipCenter 0: VME will add 0 to the user-input X value. For SkipCenter 1: VME will add 32 to the user-input X value. For SkipCenter 2: VME will add 0 to the user-input X value. For SkipCenter 3: VME will add 32 to the user-input X value. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] For chroma skip: Format = S12.3 (2's comp) Hardware Range: [-1024.000 to 1023.875]
WX+0.6	31:0	**Ref0 SkipCenter 3 Delta XY***(for definition see M3.7)*
WX+0.5	31:0	**Ref1 SkipCenter 2 Delta XY** *(for definition see M3.7)*
WX+0.4	31:0	**Ref0 SkipCenter 2 Delta XY** *(for definition see M3.7)*
WX+0.3	31:0	**Ref1 SkipCenter 1 Delta XY** *(for definition see M3.7)*
WX+0.2	31:0	**Ref0 SkipCenter 1 Delta XY** *(for definition see M3.7)*
WX+0.1	31:0	**Ref1 SkipCenter 0 Delta XY** *(for definition see M3.7)*
WX+0.0	31:0	**Ref0 SkipCenter 0 Delta XY** *(for definition see M3.7)*
WX+1.7	31:0	**Neighbor pixel Luma value [23, -1] to [20, -1].** Upper-right pixels from neighbor macroblock C
WX+1.6	31:0	**Neighbor pixel Luma value [19, -1] to [16, -1].** Upper-right edge pixels from neighbor macroblock C
WX+1.5	31:0	**Neighbor pixel Luma value [15, -1] to [12, -1].** Top edge pixels from neighbor macroblock B
WX+1.4	31:0	**Neighbor pixel Luma value [11, -1] to [8, -1].** Top edge pixels from neighbor macroblock B
WX+1.3	31:0	**Neighbor pixel Luma value [7, -1] to [4, -1].** Top edge pixels from neighbor macroblock B
WX+1.2	31:24	**Neighbor pixel Luma value [3, -1].** Fourth top edge pixel from neighbor macroblock B
	23:16	**Neighbor pixel Luma value [2, -1].** Third top edge pixel from neighbor macroblock B
	15:8	**Neighbor pixel Luma value [1, -1].** Second top edge pixel from neighbor macroblock B
	7:0	**Neighbor pixel Luma value [0, -1].** First top edge pixel from neighbor macroblock B
WX+1.1	31:24	**Corner Neighbor pixel 0.** Its content depends on IntraCornerSwap field. It swaps with Corner Neighbor pixel 1.
	23:10	Reserved: MBZ
	9:8	**Intra Compute Type (IntraComputeType)**

DWord	Bits	Name
		This field specifies the pixel components measured for intra prediction. 00: Luma + Chroma enabled 01: Luma only 1X: Intra disabled
	7:4	**AVC Intra Chroma Mode Mask (IntraChromaModeMask)** The following mask disables the chroma intra modes from the output. xxx1: VERT xx1x: HORZ x1xx: DC 1xxx: PLANAR
	3:0	**AVC Intra 16x16 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. xxx1: xx1x: x1xx: 1xxx:
WX+1.0	31:25	Reserved: MBZ
	24:16	**AVC Intra 8x8 Mode Mask (Intra16x16ModeMask):** Disables given intra mode as follows. x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:
	15:9	Reserved: MBZ
	8:0	**AVC Intra 4x4 Mode Mask (Intra16x16ModeMask):**

DWord	Bits	Name
		Disables given intra mode as follows. x xxxx xxx1: x xxxx xx1x: x xxxx x1xx: x xxxx 1xxx: x xxx1 xxxx: x xx1x xxxx: x x1xx xxxx: x 1xxx xxxx: 1 xxxx xxxx:
WX+2.7	31:24	Reserved: MBZ
	23:16	Penalty for Intra4x4 non-DC prediction mode Format: U8
	15:8	Penalty for Intra8x8 non-DC prediction mode Format: U8
	7:0	Penalty for Intra16x16 non-DC prediction mode Format: U8
WX+2.6	31:0	Reserved: MBZ
WX+2.5	31:16	Reserved: MBZ
	15:0	**Neighbor pixel Chroma value CbCr pair [-1, -1]** Corner neighbor pixel pair (CbCr pair, each U8).
WX+2.4	31:28	**Intra Predictor Mode for Neighbor B15 (IntraMxMPredModeB15):** This field carries the intra prediction mode of the fourth bottom 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	27:24	**Intra Predictor Mode for Neighbor B14 (IntraMxMPredModeB14):** This field carries the intra prediction mode of the third bottom 4x4 block (Block 14 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	23:20	**Intra Predictor Mode for Neighbor B11 (IntraMxMPredModeB11):** This field carries the intra prediction mode of the second bottom 4x4 block (Block 11 in Numbers of Block4x4 in a 16x16 region) of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	19:16	**Intra Predictor Mode for Neighbor B10 (IntraMxMPredModeB10):** This field carries the intra

DWord	Bits	Name
		prediction mode of the first bottom 4x4 block (Block 10 in Numbers of Block4x4 in a 16x16 region)of the top neighbor macroblock B. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	15:12	**Intra Predictor Mode for Neighbor A15 (IntraMxMPredModeA15):** This field carries the intra prediction mode of the fourth rightmost 4x4 block (Block 15 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	11:8	**Intra Predictor Mode for Neighbor A13 (IntraMxMPredModeA13):** This field carries the intra prediction mode of the third rightmost 4x4 block (Block 13 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification.
	7:4	**Intra Predictor Mode for Neighbor A7 (IntraMxMPredModeA7):** This field carries the intra prediction mode of the second rightmost 4x4 block (Block 7 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A.
	3:0	**Intra Predictor Mode for Neighbor A5 (IntraMxMPredModeA5):** This field carries the intra prediction mode of the first rightmost 4x4 block (Block 5 in Numbers of Block4x4 in a 16x16 region) of the left neighbor A. Definition of the term is according to Sections 8.3.1 and 8.3.2 of the AVC specification. Intra Predictor Modes for Neighbor A and B are only used if **MODE_INTRA_NOPRED** is not zero. For intra mode selection, bias is applied to the predicted mode if a predictor is present for a partition. This is achieved by applying a penalty term MODE_INTRA_NONPRED defined in the VME state to the cost functions for non-predicted modes. The predictor for a given partition is from its left neighbor and top neighbor. The intra decision for a partition serves as the predictor for the next partition in the partition order as defined in Numbers of Block4x4 in a 16x16 region and Numbers of Block4x4 in an 8x8 region or numbers of Block8x8 in a 16x16 region. This set of intra predictor mode for neighbor macroblocks are only used for INTRA8x8 and INTRA4x4 modes. Format : U4 (The value of this field is defined in Definition of Intra4x4PredMode which is the same as that in Definition of Intra8x8PredMode.)
WX+2.3	31:24	**Corner Neighbor pixel 1.** Its content depends on **IntraCornerSwap** field. It swaps with **Corner Neighbor pixel 0**. **Neighbor pixel Luma value [-1, -1]**. The one upper-left edge pixel from neighbor macroblock D, which is the right most edge pixel of D, if **IntraCornerSwap** field is **1**. Or **Neighbor pixel Luma value [-1, 15]**. The last left edge pixel from neighbor macroblock A, which is the left most edge pixel of D, if **IntraCornerSwap** field is 0.
	23:0	**Neighbor pixel Luma value [-1, 14] to [-1, 12].** Left edge pixels from neighbor macroblock A
WX+2.2	31:0	**Neighbor pixel Luma value [-1, 11] to [-1, 8].** Left edge pixels from neighbor macroblock A
WX+2.1	31:0	**Neighbor pixel Luma value [-1, 7] to [-1, 4].** Left edge pixels from neighbor macroblock A

DWord	Bits	Name
WX+2.0	31:24	**Neighbor pixel Luma value [-1, 3].** Fourth left edge pixel from neighbor macroblock A
	23:16	**Neighbor pixel Luma value [-1, 2].** Third left edge pixel from neighbor macroblock A
	15:8	**Neighbor pixel Luma value [-1, 1].** Second left edge pixel from neighbor macroblock A
	7:0	**Neighbor pixel Luma value [-1, 0].** First left edge pixel from neighbor macroblock A
WX+3.7	31:0	**Neighbor pixel Chroma value CbCr pair [7, -1] to [6, -1]**
WX+3.6	31:0	**Neighbor pixel Chroma value CbCr pair [5, -1] to [4, -1]**
WX+3.5	31:0	**Neighbor pixel Chroma value CbCr pair [3, -1] to [2, -1]**
WX+3.4	31:0	**Neighbor pixel Chroma value CbCr pair [1, -1] to [0, -1]**
WX+3.3	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 7] to [-1, 6]**
WX+3.2	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 5] to [-1, 4]**
WX+3.1	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 3] to [-1, 2]**
WX+3.0	31:0	**Neighbor pixel Chroma value CbCr pair [-1, 1] to [-1, 0]**

IME Input Message Phases

Major changes:

- Addition of the search path, no longer accessed via LUT, will come in message payload.
- Streamin\streamout now contains the 9 major shape reference indices per direction.
- Distortion precisions increased to 16b.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

Project	Value	Any Description
CHV, BSW	4	CHV, BSW added 1 additional universal message phase.

DWord	Bits	Name
WX+0.7	31:0	**IME Search Path Delta 28-31**
WX+0.6	31:0	**IME Search Path Delta 24-27**
WX+0.5	31:0	**IME Search Path Delta 20-23**
WX+0.4	31:0	**IME Search Path Delta 16-19**
WX+0.3	31:0	**IME Search Path Delta 12-15**
WX+0.2	31:0	**IME Search Path Delta 8-11**
WX+0.1	31:0	**IME Search Path Delta 4-7**
WX+0.0	31:0	**IME Search Path Delta 0-3** [7:4] (Y) – specifies relative Y distance to the next SU from previous SU in units of SU.

DWord	Bits	Name
		[3:0] (X) – specifies relative X distance to the next SU from previous SU in units of SU. Format = U8
WX+1.7	31:0	Reserved MBZ
WX+1.6	31:0	Reserved MBZ
WX+1.5	31:0	**IME Search Path Delta 52-55**
WX+1.4	31:0	**IME Search Path Delta 48-51**
WX+1.3	31:0	**IME Search Path Delta 44-47**
WX+1.2	31:0	**IME Search Path Delta 40-43**
WX+1.1	31:0	**IME Search Path Delta 36-39**
WX+1.0	31:0	**IME Search Path Delta 32-35**
WX+2.7	31:0	Reserved MBZ
WX+2.6	31:28	Rec0 Shape 8x8_3 RefID
	27:24	Rec0 Shape 8x8_2 RefID
	23:20	Rec0 Shape 8x8_1 RefID
	19:16	Rec0 Shape 8x8_0 RefID
	15:12	Rec0 Shape 8x16_1 RefID
	11:8	Rec0 Shape 8x16_0 RefID
	7:4	Rec0 Shape 16x8_1 RefID
	3:0	Rec0 Shape 16x8_0 RefID Format = U4
WX+2.5	31:16	Rec0 Shape 16x16 Y (relative to source MB)
	15:0	Rec0 Shape 16x16 X (relative to source MB)
WX+2.4	31:20	Reserved MBZ
	19:16	Rec0 Shape 16x16 RefID Format = U4
	15:0	Rec0 Shape 16x16 Distortion Format = U16
WX+2.3	31:16	Rec0 Shape 8x8_3 Distortion Format = U16 Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).

DWord	Bits	Name
	15:0	Rec0 Shape 8x8_2 Distortion Format = U16
WX+2.2	31:16	Rec0 Shape 8x8_1 Distortion Format = U16
	15:0	Rec0 Shape 8x8_0 Distortion Format = U16
WX+2.1	31:16	Rec0 Shape 8x16_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec0 Shape 8x16_0 Distortion Format = U16
WX+2.0	31:16	Rec0 Shape 16x8_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec0 Shape 16x8_0 Distortion Format = U16
WX+3.7	31:16	Rec0 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_3 X (relative to source MB)
WX+3.6	31:16	Rec0 Shape 8x8_2 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_2 X (relative to source MB)
WX+3.5	31:16	Rec0 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_1 X (relative to source MB)
WX+3.4	31:16	Rec0 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x8_0 X (relative to source MB)
WX+3.3	31:16	Rec0 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_1 X (relative to source MB)
WX+3.2	31:16	Rec0 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec0 Shape 8x16_0 X (relative to source MB)
WX+3.1	31:16	Rec0 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_1 X (relative to source MB)

DWord	Bits	Name
WX+3.0	31:16	Rec0 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec0 Shape 16x8_0 X (relative to source MB)
WX+4.7	31:0	Reserved MBZ
WX+4.6	31:28	Rec1 Shape 8x8_3 RefID
	27:24	Rec1 Shape 8x8_2 RefID
	23:20	Rec1 Shape 8x8_1 RefID
	19:16	Rec1 Shape 8x8_0 RefID
	15:12	Rec1 Shape 8x16_1 RefID
	11:8	Rec1 Shape 8x16_0 RefID
	7:4	Rec1 Shape 16x8_1 RefID
	3:0	Rec1 Shape 16x8_0 RefID Format = U4
WX+4.5	31:16	Rec1 Shape 16x16 Y (relative to source MB)
	15:0	Rec1 Shape 16x16 X (relative to source MB)
WX+4.4	31:20	Reserved MBZ
	19:16	Rec1 Shape 16x16 RefID Format = U4
	15:0	Rec1 Shape 16x16 Distortion Format = U16
WX+4.3	31:16	Rec1 Shape 8x8_3 Distortion Format = U16 Hardware only uses 14 bits. Upper bits ignored (True for all 8x8_X Distortions).
	15:0	Rec1 Shape 8x8_2 Distortion Format = U16
WX+4.2	31:16	Rec1 Shape 8x8_1 Distortion Format = U16
	15:0	Rec1 Shape 8x8_0 Distortion Format = U16

DWord	Bits	Name
WX+4.1	31:16	Rec1 Shape 8x16_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 8x16_X Distortions).
	15:0	Rec1 Shape 8x16_0 Distortion Format = U16
WX+4.0	31:16	Rec1 Shape 16x8_1 Distortion Format = U16 Hardware only uses 15 bits. Upper bits ignored (True for all 16x8_X Distortions).
	15:0	Rec1 Shape 16x8_0 Distortion Format = U16
WX+5.7	31:16	Rec1 Shape 8x8_3 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_3 X (relative to source MB)
WX+5.6	31:16	Rec1 Shape 8x8_2 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_2 X (relative to source MB)
WX+5.5	31:16	Rec1 Shape 8x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_1 X (relative to source MB)
WX+5.4	31:16	Rec1 Shape 8x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x8_0 X (relative to source MB)
WX+5.3	31:16	Rec1 Shape 8x16_1 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_1 X (relative to source MB)
WX+5.2	31:16	Rec1 Shape 8x16_0 Y (relative to source MB)
	15:0	Rec1 Shape 8x16_0 X (relative to source MB)
WX+5.1	31:16	Rec1 Shape 16x8_1 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_1 X (relative to source MB)
WX+5.0	31:16	Rec1 Shape 16x8_0 Y (relative to source MB)
	15:0	Rec1 Shape 16x8_0 X (relative to source MB)

FBR Input Message Phases

Major changes:

- Consists of the 32 sub-block motion vectors following the same 32MV format as the rest of VME.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

Project	Value	Any Description
CHV, BSW	4	CHV, BSW added 1 additional universal message phase.

DWord	Bits	Name
WX+0.7	31:0	**Ref1 Sub-block XY 3**
WX+0.6	31:0	**Ref0 Sub-block XY 3**
WX+0.5	31:0	**Ref1 Sub-block XY 2**
WX+0.4	31:0	**Ref0 Sub-block XY 2**
WX+0.3	31:0	**Ref1 Sub-block XY 1**
WX+0.2	31:0	**Ref0 Sub-block XY 1**
WX+0.1	31:0	**Ref1 Sub-block XY 0**
WX+0.0	31:16	**Ref0 Sub-block Y 0** The y-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. **Note:** All MVs must be replicated for each shape. (e.g. for luma 16x16 shape and chroma 8x8, all Sub-block MVs must be the same. For luma 8x8 shape and chroma 4x4, each 8x8 must have its respective Sub-block MVs be replicated). Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
	15:0	**Ref0 Sub-block X 0** The x-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. **Note**: All MVs must be replicated for each shape. (e.g. for luma 16x16 shape and chroma 8x8, all Sub-block MVs must be the same. For luma 8x8 shape and chroma 4x4, each 8x8 must have its respective Sub-block MVs be replicated).

DWord	Bits	Name
		Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75]
WX+1.7	31:0	**Ref1 Sub-block XY 7**
WX+1.6	31:0	**Ref0 Sub-block XY 7**
WX+1.5	31:0	**Ref1 Sub-block XY 6**
WX+1.4	31:0	**Ref0 Sub-block XY 6**
WX+1.3	31:0	**Ref1 Sub-block XY 5**
WX+1.2	31:0	**Ref0 Sub-block XY 5**
WX+1.1	31:0	**Ref1 Sub-block XY 4**
WX+1.0	31:0	**Ref0 Sub-block XY 4**
WX+2.7	31:0	**Ref1 Sub-block XY 11**
WX+2.6	31:0	**Ref0 Sub-block XY 11**
WX+2.5	31:0	**Ref1 Sub-block XY 10**
WX+2.4	31:0	**Ref0 Sub-block XY 10**
WX+2.3	31:0	**Ref1 Sub-block XY 9**
WX+2.2	31:0	**Ref0 Sub-block XY 9**
WX+2.1	31:0	**Ref1 Sub-block XY 8**
WX+2.0	31:0	**Ref0 Sub-block XY 8**
WX+3.7	31:0	**Ref1 Sub-block XY 15**
WX+3.6	31:0	Ref0 Sub-block XY 15
WX+3.5	31:0	Ref1 Sub-block XY 14
WX+3.4	31:0	Ref0 Sub-block XY 14
WX+3.3	31:0	Ref1 Sub-block XY 13
WX+3.2	31:0	Ref0 Sub-block XY 13

DWord	Bits	Name
WX+3.1	31:0	Ref1 Sub-block XY 12
WX+3.0	31:0	Ref0 Sub-block XY 12

IDM Input Message Phases

Major changes:

- Consists of the 256b source pixel mask.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware will ignore these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

"X" in "WX+..." below is:

Project	Value	Any Description
CHV, BSW	4	CHV, BSW added 1 additional universal message phase.

DWord	Bits	Name
MX+0.7	31:0	**Source MB Pixel Mask Row 14, 15**
MX+0.6	31:0	**Source MB Pixel Mask Row 12, 13**
MX+0.5	31:0	**Source MB Pixel Mask Row 10, 11**
MX+0.4	31:0	**Source MB Pixel Mask Row 8, 9**
MX+0.3	31:0	**Source MB Pixel Mask Row 6, 7**
MX+0.2	31:0	**Source MB Pixel Mask Row 4, 5**
MX+0.1	31:0	**Source MB Pixel Mask Row 2, 3**
MX+0.0	31:16	**Source MB Pixel Mask Row 1**
	15:0	**SourceMB Pixel Mask Row 0** These fields disable a given pixel of the 16x16 source MB. They are arranged from left to right with bit 0 being the left-most pixel and bit 15 being the right-most pixel of the source MB. Format = disable

Return Data Message Phases

Major changes:

Major Change
Many of the fields are not valid output for all message types.
Addition of new message phase, which has the block reference IDs and forward transform skip check data.
Intra chroma distortion and best mode are reported.
All U14 distortion values are now U16.

ValidMsgType = "..." identifies the given field is required for each message type. Hardware ignores these fields under messages where that field is invalid. Hardware output for non valid fields is undefined.

DWord	Bits	Description
W0.7	31:16	**Total VME Stalled Clocks:** Counts the number of clocks VME is stalled or starved while processing this request, due to cache misses. Format: U16 ValidMsgType = SIC, IME, FBR
	15:0	**Total VME Compute Clocks:** Counts the number of clocks VME is processing this request, but not stalled or starved as a result of cache misses. Format: U16 ValidMsgType = SIC, IME, FBR
W0.6	31:26	**Alternate Search Path Length:** Counts the number of unique search units computed by VME for the alternate search path for dual reference or dual search path. If the search path would return to a previously processed SU, it would not be reprocessed and hence not recounted. The value of [W0.1 15:8] is the overall total search units processed from both paths whereas this value is the contribution only from the second search path. **Note:** Whenever VME is in a mode that processes only a single search path, this field is 0x0. Format: U6, Range of 0-48 ValidMsgType = IME
	25	**MaxMV Occurred:** This bit is set if the MaxMV event prevented the lowest distortion solution is rejected due to lack of motion vectors. Format: U1 Valid only for Luma Source Size = 16x16. ValidMsgType = IME
	24	**EarlyIMEStop Occurred:**

DWord	Bits	Description
		This bit is set if the EarlyIMEStop threshold is satisfied and IME discontinues searching. Format: U1 ValidMsgType = IME
	23:16	**Sub-Macroblock Prediction Mode (SubMbPredMode):** If **InterMbMode** is INTER8x8, this field describes the prediction mode of the sub-partitions in the four 8x8 sub-macroblock. It contains four subfields each with 2 bits, corresponding to the four 8x8 sub-macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 macroblock. This field is derived from **MbType** for a non-BP_8x8 inter macroblock, and carries redundant information as **MbType**. If **InterMbMode** is INTER16x16, INTER16x8 or INTER8x16, this field carries the prediction modes of the sub macroblock (one 16x16, two 16x8, or two 8x16). The unused bits are set to zero. Bits [1:0]: SubMbPredMode[0] Bits [3:2]: SubMbPredMode[1] Bits [5:4]: SubMbPredMode[2] Bits [7:6]: SubMbPredMode[3] ValidMsgType = SIC, IME, FBR
	15:8	**Sub-Macroblock Shape (SubMbShape):** This field describes the subdivision of the four 8x8 sub-macroblocks. It contains four subfields each with 2 bits, corresponding to the four 8x8 sub macroblocks in sequential order. This field is derived from sub_mb_type for a BP_8x8 or equivalent macroblock. This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as **MbType**. This field is only valid if **InterMbMode** is INTER8x8; otherwise, it is set to zero. Bits [1:0]: SubMbShape[0] Bits [3:2]: SubMbShape[1] Bits [5:4]: SubMbShape[2] Bits [7:6]: SubMbShape[3] ValidMsgType = SIC, IME, FBR
	7:0	**Macroblock Intra Structure (MbIntraStruct):** This is a bitmask that specifies neighbor macroblock availability. This allows software to constrain intra prediction mode search. This field is simply copied from the input message (to reduce software overhead of forming the output message to PAK).

DWord	Bits	Description
		<table><tr><th>Bits</th><th>MotionVerticalFieldSelect Index</th></tr><tr><td>7</td><td>**Reserved: MBZ (for IntraPredAvailFlagF – F** (pixel[-1,7] available for MbAff)</td></tr><tr><td>6</td><td>**Reserved: MBZ (for IntraPredAvailFlagA/E – A** (left neighbor top half for MbAff)</td></tr><tr><td>5</td><td>**IntraPredAvailFlagE/A – A** (Left neighbor or Left bottom half)</td></tr><tr><td>4</td><td>**IntraPredAvailFlagB – B** (Upper neighbor)</td></tr><tr><td>3</td><td>**IntraPredAvailFlagC – C** (Upper left neighbor)</td></tr><tr><td>2</td><td>**IntraPredAvailFlagD – D** (Upper right neighbor)</td></tr><tr><td>1:0</td><td>**ChromaIntraPredMode**</td></tr></table> **Note:** This 8b field is MBZ when IntraComputeType == 1X (when intra is disabled). ValidMsgType = SIC
W0.5	31:16	**LumaIntraPredModes[3]**
	15:0	**LumaIntraPredModes[2]**
W0.4	31:16	**LumaIntraPredModes[1]**
	15:0	**LumaIntraPredModes[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block (Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes. 4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes. **Note:** The LumaIntraPredModes are MBZ when IntraComputeType == 1X (when intra is disabled). ValidMsgType = SIC
W0.3	31:16	**BestChromaIntraDistortion** This field provides the ChromaIntraMode distortion (sum of Cb and Cr dist). Note: this field is MBZ when IntraComputeType == 1X (when intra is disabled). Format = U16 ValidMsgType = SIC
	15:0	**BestIntraDistortion** This field provides redundant information. The IntraMbMode indicates if this is a 16x16/8x8/4x4 distortion. **Note:** This field is MBZ when IntraComputeType == 1X (when intra is disabled).

DWord	Bits	Description
		Format = U16 ValidMsgType = SIC
W0.2	31:16	**SkipRawDistortion** This field contains Skip Raw Distortion which may be used by software to further refine the skip decision. **Note:** this field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
	15:0	**InterDistortion** This field provides the best inter distortion. Format = U16 ValidMsgType = SIC, IME, FBR
W0.1	31:27	Reserved: MBZ
	26:16	**Sum Ref1 Inter Dist Upper 10 bits (SumInterDistL1Upper)** Contains the sum distortion of all 16x16 Inter shape of Ref1 within the searched SU of this search window. MSB 11 bits only. Format = U11 ValidMsgType = IME
	15:8	**Search Path Length:** This field returns the number of SU it takes in the integer search. It includes predetermined search path and dynamic search path. Format: U8 ValidMsgType = IME
	7:4	**Reference 1 border reached:** Bitmask indicating whether any border of reference 1 is reached by one or more motion vectors in the winning inter mode. xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached ValidMsgType = IME
	3:0	**Reference 0 border reached:** Bitmask indicating whether any border of reference 0 is reached by one or more motion vectors in the winning inter mode.

DWord	Bits	Description
		xxx1: left border reached xx1x: right border reached x1xx: top border reached 1xxx: bottom border reached ValidMsgType = IME
W0.0	31:29	Reserved: MBZ
	28:24	**MvQuantity:** Specifies the number of MVs in packed format (in units of motion vectors). **Note:** This field is provided to help software meet conformance requirements such as maximum number of motion vectors for two consecutive macroblocks. Format: U5, valid from 0 to 32 ValidMsgType = SIC, IME, FBR
	23	**ExtendedForm:** This field specifies that **LumaIntraMode**'s are fully replicated in 4x4 sub-blocks respectively. And motion vectors must be in unpacked form as well. This non-DXVA form is used for optimal kernel performance. **This is reserved MBZ and the HW always extends.** ValidMsgType = SIC
	22:21	Reserved: MBZ
	20:16	**IntraMbType** This field is encoded to match with the inter type determined as described in the next section. It follows a unified encoding for intra macroblocks according to the AVC Spec. **Note:** This field is MBZ when IntraComputeType == 1X (when intra is disabled). ValidMsgType = SIC
	15	**Transform8x8Flag (Transform 8x8 Flag)** This field indicates that 8x8 transform is recommended. It is set to 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8. For **IntraMbFlag** = INTER. If **T8x8FlagForInterEn** = 0, this field is set to 0 by VME hardware. If **T8x8FlagForInterEn** = 1, this field is set to 1 if there is no sub macroblock size less than 8x8 (noSubMbPartSizeLessThan8x8Flag = 1). 0: 4x4 integer transform 1: 8x8 integer transform **Note:** This bit is always 0 for non-16x16 source block cases. ValidMsgType = IME, FBR

DWord	Bits	Description
	14	**FieldMbFlag** This field indicates the inter prediction result is field or frame. It is always set to **SrcAccess**. 0: Frame macroblock 1: Field macroblock ValidMsgType = SIC, IME, FBR
	13	Reserved: MBZ
	12:8	**InterMbType** This field is encoded to match with the inter type determined as described in the next section. It follows a unified encoding for inter macroblocks according to AVC Spec. ValidMsgType = SIC, IME, FBR
	7	**FieldMbPolarityFlag** This field indicates the field polarity of the current macroblock. Unique for AVC standard, within an MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in an MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0. Within a field picture in most coding standard, this field is a constant for the whole field picture. It is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. This field is reserved and MBZ for a progressive frame picture. VME hardware set this field to 1 if the source block is a field block from the bottom field and otherwise sets it to 0. This is accomplished by the following equation using input signals **SrcAccess** and **SrcY**: **SrcAccess** && (bit0(**SrcY**) ==1). 0 = Current macroblock is a field macroblock from the **top** field. 1 = Current macroblock is a field macroblock from the **bottom** field. Equals SrcAccess && SrcFieldPolarity(M1.7[19]) ValidMsgType = SIC, IME, FBR
	6	Reserved: MBZ
	5:4	**IntraMbMode** This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if intra mode is enabled. ValidMsgType = SIC

DWord	Bits	Description
	3:2	Reserved: MBZ
	1:0	**InterMbMode** This field is provided to carry redundant information as that in **MbType**. The full extended definition of this field allows kernel software to help update the **MbType** field when outputting controls to the MFX PAK encoding. VME outputs this field regardless of MbIntraFlag value if inter mode is enabled. ValidMsgType = SIC, IME, FBR
W1.7 to W1.2	31:0 Each	**MVb[3]** to **MVb[1]**. Motion vectors 3 to 1 for Reference 1, and **MVa[3]** to **MVa[1]**. Motion vectors 3 to 1 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W1.1	31:16	**MVb[0].y:** Returning the y-coordinate of Motion Vector 0 for Reference 1, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
	15:0	**MVb[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 1, relative to source MB location. Its meaning is determined by **MbType**. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W1.0	31:16	**MVa[0].y:** Returning the y-coordinate of Motion Vector 0 for Reference 0, relative to source MB location. Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
	15:0	**MVa[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ the first pixel in 6 by 2 block) for Reference 0, relative to source MB location. Its meaning is determined by **MbType**. Hardware Range: [-2048.00 to 2047.75]

DWord	Bits	Description
		ValidMsgType = IME **MVa[0].x:** Returning the x-coordinate of Motion Vector 0 (co-located w/ sublbock_4x4_0) for Reference 0, relative to source MB location. Its meaning is determined by **MbType**. The returned motion vectors are placed in a fixed data format, with up to 16 motion vectors for one reference and the motion vectors from reference 0 and 1 interleaved. If M1.7:20 (Enable WeightedSAD) is set true (1), then this field and the rest of W1 through W4 are redefined as follows: W1.0 from 31:0 = Minimum 16x16 distortion when applying 1st weighting pattern (16x16_0) W1.1 from 31:0 = Reserved W1.2 from 31:0 = Minimum 16x16 distortion when applying 2nd weighting pattern (16x16_1) W1.3 from 31:0 = Reserved W1.4 from 31:0 = Minimum 16x16 distortion when applying 3rd weighting pattern (16x16_2) W1.5 from 31:0 = Reserved W1.6 from 31:0 = Minimum 16x16 distortion when applying 4th weighting pattern (16x16_3) W1.7 from 31:0 = Reserved W2 through W4 = Reserved Format = S13.2 (2's comp) Hardware Range: [-2048.00 to 2047.75] ValidMsgType = SIC, IME, FBR
W2.7 to W2.0	31:0 Each	**MVb[7]** to **MVb[4]**. Motion vectors 7 to 4 for Reference 1, and **MVa[7]** to **MVa[4]**. Motion vectors 7 to 4 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W3.7 to W3.0	31:0 Each	**MVb[11]** to **MVb[8]**. Motion vectors 11 to 8 for Reference 1, and **MVa[11]** to **MVa[8]**. Motion vectors 11 to 8 for Reference 0 See note in W1.0 bits 15:0.

DWord	Bits	Description
		ValidMsgType = SIC, IME, FBR
W4.7 to W4.0	31:0 Each	**MVb[15]** to **MVb[12]**. Motion vectors 15 to 12 for Reference 1, and **MVa[15]** to **MVa[12]**. Motion vectors 15 to 12 for Reference 0 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W5.7 to W5.1	31:0 Each	**InterDistortion[15] to InterDistortion[2]**. Inter-prediction-distortion associated with motion vector 15 to 2. Its meaning is determined by sub-shape. See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
W5.0	31:16	**InterDistortion[1].** Inter-prediction-distortion with motion vector 1 (co-located with subblock_4x4_1). Its meaning is determined by sub-shape. Format = U16 See note in W1.0 bits 15:0. ValidMsgType = SIC, IME, FBR
	15:0	**InterDistortion[0].** Inter-prediction-distortion associated with motion vector 0 (co-located with subblock_4x4_0). Its meaning is determined by sub-shape. It must be zero if the corresponding sub-shape is not chosen. This field may be associated with MVa[0] and/or MVb[0], depending on the resulting prediction mode for the sub-block. If the corresponding MV field is created by "duplication", this field must be zero. For 1MVP skip messages, the 16x16 distortion (sad + mv cost + ref cost) is present here. For 4MVP skip messages, the 4 8x8 distortions (sad + mv cost + ref cost) are present here. **Note:** This set of inter-prediction-distortion fields contains detailed information for all sub-shapes. It may be used to assist a multi-pass motion search algorithm. The overall distortion for the macroblock is provided in W0.1. **Format = U16** If M1.7:20 (Enable WeightedSAD) is set true (1), then this field and the rest of W5 are redefined as follows: W5.0 from 15:0 = Minimum 16x16 distortion when applying 1st weighting pattern (16x16_0

DWord	Bits	Description
		with corresponding MV at W1.0) W5.0 from 31:16 = Minimum 16x16 distortion when applying 2nd weighting pattern (16x16_1 with corresponding MV at W1.2) W5.1 from 15:0 = Minimum 16x16 distortion when applying 3rd weighting pattern (16x16_2 with corresponding MV at W1.4) W5.1 from 31:16 = Minimum 16x16 distortion when applying 4th weighting pattern (16x16_3 with corresponding MV at W1.6) W5.7 to W5.2 = Reserved
W6.7	31:16	**Max Ref1 Inter Dist (MaxRef1InterDist)** Contains the distortion of the 16x16 Inter shape of Ref1 with the maximum distortion within the searched SU of this search window. Format = U16 ValidMsgType = IME
	15:0	**Max Ref0 Inter Dist (MaxRef0InterDist)** Contains the distortion of the 16x16 Inter shape of Ref0 with the maximum distortion within the searched SU of this search window. Format = U16 ValidMsgType = IME
W6.6	31:27	Reserved: MBZ
	26:0	**Sum Ref0 Inter Dist (SumRef0InterDist)** Contains the sum distortion of all 16x16 Inter shape of Ref0 within the searched SU of this search window. Format = U27 ValidMsgType = IME
W6.5	31:16	**Block 0 Chroma Cr Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
	15:0	**Block 0 Chroma Cb Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold.

DWord	Bits	Description
		Note: This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
	31:16	**Sum Ref1 Inter Dist lower 16 bits (SumInterDistL1Lower)** Contains the sum distortion of all 16x16 Inter shape of Ref1 within the searched SU of this search window. Lower 16 bits. Format = U16 ValidMsgType = IME
	15:8	**Block 0 Chroma Cr NZC** Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
	7:0	**Block 0 Chroma Cb NZC** Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
W6.3	31:16	**Block 3 Luma Coeff Magnitude Clip Sum**
	15:0	**Block 2 Luma Coeff Magnitude Clip Sum**
W6.2	31:16	**Block 1 Luma Coeff Magnitude Clip Sum**
	15:0	**Block 0 Luma Coeff Magnitude Clip Sum** Sum of how much all the coefficients across 1 block exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U16 ValidMsgType = SIC
W6.1	31:24	**Block 3 Luma NZC**
	23:16	**Block 2 Luma NZC**
	15:8	**Block 1 Luma NZC**
	7:0	**Block 0 Luma NZC**

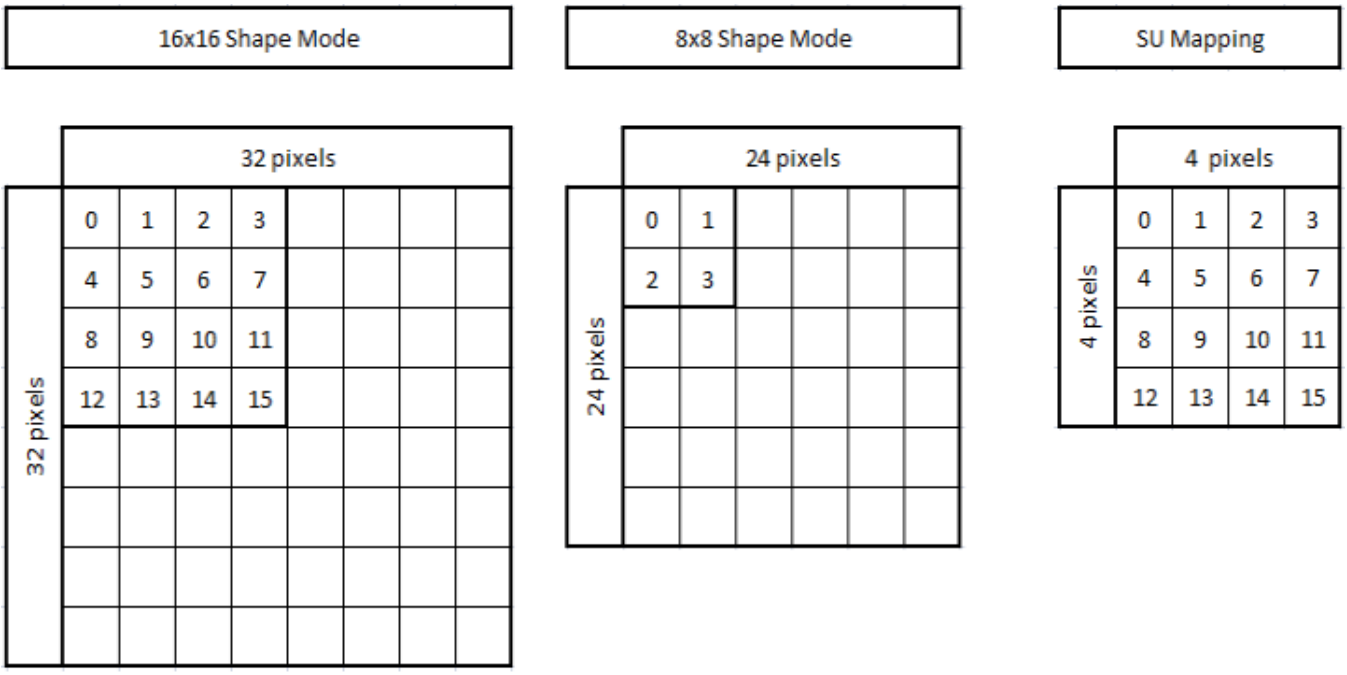
DWord	Bits	Description
		Count of the coefficients across 1 block that exceeded their respective threshold. **Note:** This field is MBZ when SkipModeEn is not set (when skip is disabled). Format = U8 ValidMsgType = SIC
W6.0	31:28	**Bwd Block 3 RefID**
	27:24	**Fwd Block 3 RefID**
	23:20	**Bwd Block 2 RefID**
	19:16	**Fwd Block 2 RefID**
	15:12	**Bwd Block 1 RefID**
	11:8	**Fwd Block 1 RefID**
	7:4	**Bwd Block 0 RefID** Reference ID for backward block 0. **Note:** Even if shape is 16x16, this field is defined per block, hence VME will replicate the RefID for larger shapes. Replication happens only in IME. For CRE (SIC/FBR), this is a pass through field. Format = U4 ValidMsgType = SIC, IME, FBR
	3:0	**Fwd Block 0 RefID** Reference ID for forward block 0. **Note:** Even if shape is 16x16, this field is defined per block, hence VME will replicate the RefID for larger shapes. Replication happens only in IME. For CRE (SIC/FBR), this is a pass through field. Format = U4 ValidMsgType = SIC, IME, FBR

IME StreamOut

IME Streamout follows the same format as the IME Streamin message phases (IME2-IME5).

IDM Stream-Out

Because the IDM output message only contains distortion values without their corresponding motion vectors, a direct mapping system is used to derive the MV associated with a given distortion.



This mapping allows the user to derive the location of a given distortion easily by applying the following equations.

ShapeMode == 16x16	ShapeMode == 8x8
MV.x = (GRF# % 4) * 4 + W# % 4 + Ref.x	**MV.x = ((GRF# / 4) % 2) * 4 + W# % 4 + Ref.x**
MV.y = (GRF# / 4) * 4 + W# / 4 + Ref.y	**MV.y = ((GRF# / 4) / 2) * 4 + W# / 4 + Ref.y**

- GRF#: The 256b return message phase number.
- W#: The 16 16b word within each 256b return message.

IDM16x16 Streamout Message Format

DWord	Bits	Description
W15	31:0	**Distortion Mesh Block 15 Search Points 0-15**
W14	31:0	**Distortion Mesh Block 14 Search Points 0-15**
W13	31:0	**Distortion Mesh Block 13 Search Points 0-15**
W12	31:0	**Distortion Mesh Block 12 Search Points 0-15**
W11	31:0	**Distortion Mesh Block 11 Search Points 0-15**
W10	31:0	**Distortion Mesh Block 10 Search Points 0-15**
W9	31:0	**Distortion Mesh Block 9 Search Points 0-15**
W8	31:0	**Distortion Mesh Block 8 Search Points 0-15**
W7	31:0	**Distortion Mesh Block 7 Search Points 0-15**
W6	31:0	**Distortion Mesh Block 6 Search Points 0-15**
W5	31:0	**Distortion Mesh Block 5 Search Points 0-15**

DWord	Bits	Description
W4	31:0	**Distortion Mesh Block 4 Search Points 0-15**
W3	31:0	**Distortion Mesh Block 3 Search Points 0-15**
W2	31:0	**Distortion Mesh Block 2 Search Points 0-15**
W1	31:0	**Distortion Mesh Block 1 Search Points 0-15**
W0.7	31:16	**Distortion Mesh Block 0 Search Point 15**
	15:0	**Distortion Mesh Block 0 Search Point 14**
W0.6	31:16	**Distortion Mesh Block 0 Search Point 13**
	15:0	**Distortion Mesh Block 0 Search Point 12**
W0.5	31:16	**Distortion Mesh Block 0 Search Point 11**
	15:0	**Distortion Mesh Block 0 Search Point 10**
W0.4	31:16	**Distortion Mesh Block 0 Search Point 9**
	15:0	**Distortion Mesh Block 0 Search Point 8**
W0.3	31:16	**Distortion Mesh Block 0 Search Point 7**
	15:0	**Distortion Mesh Block 0 Search Point 6**
W0.2	31:16	**Distortion Mesh Block 0 Search Point 5**
	15:0	**Distortion Mesh Block 0 Search Point 4**
W0.1	31:16	**Distortion Mesh Block 0 Search Point 3**
	15:0	**Distortion Mesh Block 0 Search Point 2**
W0.0	31:16	**Distortion Mesh Block 0 Search Point 1**
	15:0	**Distortion Mesh Block 0 Search Point 0** This is the distortion value at the location (0,0) within the search window. Each 256b (8 DW) message phase contains 16 distortion values for a given 4x4 search unit of the search window. Each word of a message phase contains a single 16x16 distortion at a given search point. The individual distortion values are arranged in raster-scan (row-major order) within the 4x4. For ShapeMode == 16, the 16 message phases comprise the 16 SUs (4x4) of the 32x32 search window for each 16x16 distortion and are returned in raster-scan (row-major order). Format = U16

IDM8x8 Streamout Message Format

DWord	Bits	Description
W15	31:0	**Distortion Mesh Block 3 SU3 Search Points 0-15**
W14	31:0	**Distortion Mesh Block 2 SU3 Search Points 0-15**
W13	31:0	**Distortion Mesh Block 1 SU3 Search Points 0-15**
W12	31:0	**Distortion Mesh Block 0 SU3 Search Points 0-15**

DWord	Bits	Description
W11	31:0	**Distortion Mesh Block 3 SU2 Search Points 0-15**
W10	31:0	**Distortion Mesh Block 2 SU2 Search Points 0-15**
W9	31:0	**Distortion Mesh Block 1 SU2 Search Points 0-15**
W8	31:0	**Distortion Mesh Block 0 SU2 Search Points 0-15**
W7	31:0	**Distortion Mesh Block 3 SU1 Search Points 0-15**
W6	31:0	**Distortion Mesh Block 2 SU1 Search Points 0-15**
W5	31:0	**Distortion Mesh Block 1 SU1 Search Points 0-15**
W4	31:0	**Distortion Mesh Block 0 SU1 Search Points 0-15**
W3	31:0	**Distortion Mesh Block 3 SU0 Search Points 0-15**
W2	31:0	**Distortion Mesh Block 2 SU0 Search Points 0-15**
W1	31:0	**Distortion Mesh Block 1 SU0 Search Points 0-15**
W0.7	31:16	**Distortion Mesh Block 0 SU0 Search Point 15**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 14**
W0.6	31:16	**Distortion Mesh Block 0 SU0 Search Point 13**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 12**
W0.5	31:16	**Distortion Mesh Block 0 SU0 Search Point 11**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 10**
W0.4	31:16	**Distortion Mesh Block 0 SU0 Search Point 9**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 8**
W0.3	31:16	**Distortion Mesh Block 0 SU0 Search Point 7**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 6**
W0.2	31:16	**Distortion Mesh Block 0 SU0 Search Point 5**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 4**
W0.1	31:16	**Distortion Mesh Block 0 SU0 Search Point 3**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 2**
W0.0	31:16	**Distortion Mesh Block 0 SU0 Search Point 1**
	15:0	**Distortion Mesh Block 0 SU0 Search Point 0** This is the distortion value at the location (0,0) within the search window. Each 256b (8 DW) message phase contains 16 distortion values for a given 4x4 search unit of the search window. Each word of a message phase contains a single 8x8 distortion at a given search point. The individual distortion values are arranged in raster-scan (row-major order) within the 4x4. For ShapeMode == 8, the first 4 message phases comprise the 1st SU of the 24x24 search window for all 4 block8x8_0-block8x8_1 distortions and are returned in raster-scan (row-major order). The next 4 message phases contain the 2nd SU and so on. Format = U16

Sample_8x8 State

This section contains different state definitions.

This state definition is used only by the *deinterlace* message. This state is stored as an array of up to 8 elements, each of which contains the dwords described here. The start of each element is spaced 8 dwords apart. The first element of the array is aligned to a 32-byte boundary. The index with range 0-7 that selects which element is being used is multiplied by 2 to determine the **Sampler Index** in the message descriptor.

SURFACE_STATE for Deinterlace, sample_8x8, and VME

This section contains media surface state definitions.

MEDIA_SURFACE_STATE

Restrictions: The Faulting modes described in the MEMORY_OBJECT_CONTROL_STATE should be set to the same for the multi-surface Video Analytics functions like “LBP Correlation” and “Correlation Search” for both the surfaces.

SAMPLER_STATE for Sample_8x8 Message

SAMPLER_STATE has different formats, depending on the message type used. For CHV, BSW, the sample_8x8 and deinterlace messages use a different format of SAMPLER_STATE as detailed in the corresponding sections.

SAMPLER_STATE

- [CHV, BSW] Function AVS Scaling : State is 154DWs
- [CHV, BSW] Function Convolve : State is 144DWs
- [CHV, BSW] Function MinMaxfilter/Erode/Dilate : State is 8DWs
- [CHV, BSW] Function MinMax/BoolCentroid/Centroid : State is 0DWs. The sampler state is not required.
- [CHV, BSW]

  This state definition is used only by the *sample_8x8* message only for specific function and the length of the state varies according to the function programmed in the message header.

  For AVS and Convolve the state is stored as an array of up to 48 elements (**192** DWs), (upsize to 64 elements, 256DWs), each of which contains the dwords described here. The start of each of this state is spaced **256** dwords apart. The first element of the array is aligned to a 64-byte boundary. The **sampler index** in the message descriptor is multiplied by **32** to determine the offset from the base where the sampler state is to be read from. Sampler states with lower foot print than 32 elements should be packed at lower offsets and this sampler state for sample_8x8 message should be kept at the end. We can reuse existing sampler_index if the result of the multiplication of 32 is not overlapping with the existing states already programmed at the lower offsets. Two adjacent state of this type should have a space of 2 sample index.

  For MinMaxFilter, Erode and Dilate the state is stored as an array of up to 2 elements (8 DWs), each of which contains the dwords as described in following section. The start of each of this state is spaced **8**

dwords apart. The first element of the array is aligned to a 32-byte boundary. The **sampler index** in the message descriptor is multiplied by **2** to determine the offset from the base where the sampler state is to be read from. Sampler states with lower foot print than 2 elements should be packed at lower offsets and this sampler state for sample_8x8 message should be kept after it. Sampler states with larger footprint (192DWs) as described earlier should be packed after this. We can reuse existing sampler_index if the result of the multiplication of 2 is not overlapping with the existing states already programmed at the lower offsets. Two adjacent state of this type should have a space of 1 sample index.

SAMPLER_STATE_8x8_AVS

SAMPLER_STATE_8x8_AVS_COEFFICIENTS

SAMPLER_STATE_8x8_CONVOLVE [CHV, BSW]

SAMPLER_STATE_8x8_ERODE_DILATE_MINMAXFILTER

Media Object Dispatch Pseudocode

```
// Variables:
Frame Height in pixels => frame_height
Frame Width  in pixels => frame_width
Frame Height in Blocks => fh
Frame Width  in Blocks => fw
Block Height in Pixels => block_height = Interlaced ? 4 : 8

// Code:
fw = frame_width  / 16;
fh = frame_height / block_height;
```

Calculate Residual Blocks Pseudocode

```
If ( fh % (2**stride) ) ≠ 0 {
    Y_Blocks_Remainder = (fh % (2**stride))
    If ( Y_Blocks_Remainder > (2**stride) / 2 ) {
        Y_Blocks_Remainder_HS1 = (2**stride) / 2
        Y_Blocks_Remainder_HS2 = Y_Blocks_Remainder - (2**stride) / 2
    }
    Else {
        Y_Blocks_Remainder_HS1 = Y_Blocks_Remainder
        Y_Blocks_Remainder_HS2 = 0
    }
}
Else {
    Y_Blocks_Remainder_HS1 = 0
    Y_Blocks_Remainder_HS2 = 0
}
```

Dispatch Media Object Pseudocode

```
 total_media_obj_cnt = fw * fh;
 reminder_media_obj_cnt_HS1 = fw * Y_Blocks_Remainder_HS1;
 reminder_media_obj_cnt_HS2 = fw * Y_Blocks_Remainder_HS2;

 ping_pong_media_obj_cnt = total_media_obj_cnt - (reminder_media_obj_cnt_HS1 +
reminder_media_obj_cnt_HS1);

 for ( i = 0; i < ping_pong_media_obj_cnt; i++ ) {
     if ( i % 2 == 0 ) {
         dispatch_media_object_hs1;
     }
     else {
         dispatch_media_object_hs2;
     }
 }

 for ( i = 0; i < reminder_media_obj_cnt_HS1; i++ ) {
     dispatch_media_object_hs1;
 }

 for ( i = 0; i < reminder_media_obj_cnt_HS2; i++ ) {
     dispatch_media_object_hs2;
 }
```

SIMD32/64 Payload

SIMD32 Payload

DWord	Bits	Description
M1.7	31:0	**Reserved**
M1.6	31:0	**U 2nd Derivative** Defines the change in the delta U for adjacent pixels in the X direction. Format = IEEE_Float in normalized space.
M1.5	31:0	**Delta V**: defines the difference in V for adjacent pixels in the Y direction. **Programming Notes:** • **Delta V** multiplied by **Height** in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • This field is ignored for the deinterlace message type. • Negative **Delta V** are not supported and should be clamped to 0. Format = IEEE_Float in normalized space.
M1.4	31:0	**Delta U:** defines the difference in U for adjacent pixels in the X direction. **Programming Notes:** • **Delta U** multiplied by **Width** in SURFACE_STATE must be less than or equal to 3 for

DWord	Bits	Description
		sample_unorm* message types. • This field is ignored for the deinterlace message type. • Negative **Delta U** are not supported and should be clamped to 0. Format = IEEE_Float in normalized space.
M1.3	31:0	**Pixel 0 V Address** Format: sample_unorm* : IEEE_Float in normalized space. Deinterlace: U32 (Range: [0,2046]) **[CHV, BSW]:** Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
M1.2	31:0	**Pixel 0 U Address** Format: sample_unorm* : IEEE_Float in normalized space. Deinterlace: U32 (Range: [0,4095]) Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
M1.1	31:0	**Reserved**
M1.0	31:0	Ignored

SIMD64 Payload

DWord	Bits	Description
M1.7	31:28	**Functionality** Value / Function: 0000b AVS scaling 0001b Convolve 0010b MINMAX 0011b MINMAXFILTER 0100b ERODE 0101b Dilate 0110b BoolCentroid / BoolSum 0111b Centroid Other Reserved
	27	**Reserved**

DWord	Bits	Description
	26:23	**Control** **[26:25] Message_Seq:** **When Functionality is AVS:** 11: 4x4 (only supported with Shuffle_OutputWriteback=1) 10: 16x8 01: 8x4 (only supported with Shuffle_OutputWriteback=1) 00: 16x4 **When Functionality is Convolve/MinMaxFilter/LBP Correlation/LBP Creation:** 11: 1x1 – This would be used in MinMaxFilter case & 1pixel convolve only for operation in scattered regions and don't want to waste the power. This mode is not used and is reserved for 2D/1D Convolve, LBP Creation, and LBP Correlation. 10: 16x1 – This would be used in case we require the operation in scattered regions and don't want to waste the throughput. 1-D Horizontal this will be 1x16. Not supported in LBP creation and LBP correlation. 0x: 16x4 – except for 1-D Horizontal this will be 4x16; illegal for 1 pixel convolve. **When Functionality is Erode/Dilate:** 11: 32x1 – This would be used in case we require the operation in scattered regions and don't want to waste the throughput. 10: 64x1 – This would be used in case we require the operation in scattered regions and don't want to waste the throughput. 01: 32x4 00: 64x4 **When Functionality is MINMAX/BoolCentroid/Centroid** 1x: These modes should always force this to 1 since we don't have SI sequencing. 0x: Unexpected behavior **[24:23]:** **When Functionality is MinMaxFilter/MinMax/LBP Creation Only:** 11: Reserved 10: Disable Max Filter. (Min Filter only)/5x5 LBP 01: Disable Min Filter. (Max Filter only)/3x3 LBP 00: Both Min and Max filter/Both 3x3 and 5x5 LBP **When Functionality is Convolve:** **Description** 1x: Reserved

DWord	Bits	Description
		00: (default used for kernel size lesser than 15x15 size) **Reserved for all other functionality.**
	22	Reserved: MBZ
	21:0	**Group ID Number** Used to group messages for reorder for sample_8x8 messages. All messages with the same Group ID must have the following in common: SURFACE_STATE, SAMPLER_STATE, Destination register on send instruction, M0, and M1 except for **Horizontal** and **Vertical Block Number**.
M1.6	31:0	**Function = AVS:** **U 2nd Derivative:** Defines the change in the delta U for adjacent pixels in the X direction. **Programming Note** **Context:** SIM64 Payload • This field is ignored for messages other than sample_8x8. • *(64 – (2*du))/35 >= ddu >= -du/18* • 2*ddu <= du Format = IEEE_Float in normalized space If function != AVS this field is ignored and assumed to be 0 (Reserved: MBZ).
M1.5	31:0	**Function = AVS:** **Delta V:** defines the difference in V for adjacent pixels in the Y direction. **Programming Note** **Context:** SIMD64 Payload • CHV, BSW: Delta V multiplied by Height in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types. • Delta V multiplied by Height in SURFACE_STATE must be less than 16 for the sample_8x8 message type. • This field is ignored for the deinterlace message type and VA sample_8x8 messages. • Negative Delta V is not supported and should be clamped to 0. Format = IEEE_Float in normalized space If function != AVS this field is ignored and assumed to be 1.
M1.4	31:0	**Function = AVS:**

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>Delta U: defines the difference in U for adjacent pixels in the X direction.
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>SIMD64 Payload</td></tr><tr><td colspan="2">• CHV, BSW: Delta U multiplied by Width in SURFACE_STATE must be less than or equal to 3 for sample_unorm* message types.
• Delta U multiplied by Height in SURFACE_STATE must be less than 16 for the sample_8x8 message type.
• This field is ignored for the deinterlace message type and VA sample_8x8 messages.
• Negative Delta U is not supported and should be clamped to 0.</td></tr></table>
Format = IEEE_Float in normalized space
If function != AVS this field is ignored and assumed to be 1.</td></tr>
<tr><td>M1.3</td><td>31:0</td><td>Pixel 0 V Address
Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
Format:
sample_unorm* and sample_8x8: IEEE_Float in normalized space
deinterlace: U32 (Range: [0,2046])
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>SIMD64 Payload</td></tr><tr><td colspan="2">• V Address should be programming within the range 0.0 to 1.0 for sampler_8x8 VA centroid and boolcentroid functions.
• V Address should be programming within the range -1.0 to 2.0 for other sample_8x8 VA function in Mirror mode.</td></tr></table></td></tr>
<tr><td>M1.2</td><td>31:0</td><td>Pixel 0 U Address
Specifies the address for the pixel at the top left of the group and not the top of the message block sent in.
Format:
sample_unorm* and sample_8x8: IEEE_Float in normalized space
deinterlace: U32 (Range: [0,4095])
<table><tr><th colspan="2">Programming Note</th></tr><tr><th>Context:</th><td>SIMD64 Payload</td></tr><tr><td colspan="2">• V Address should be programming within the range 0.0 to 1.0 for sampler_8x8 VA</td></tr></table></td></tr>
</table>

DWord	Bits	Description
		centroid and boolcentroid functions. • V Address should be programming within the range -1.0 to 2.0 for other sample_8x8 VA function in Mirror mode.
M1.1	8:0	**Vertical Block Number** Specifies the vertical block offset for the block being sent for the sample_8x8 message. This will be equal to the vertical pixel offset from the given address pixel 0 V address in multiples of blocks as defined below. Vertical Offset: if Functionality= (1D_Horz_Convolve OR Box_Filter_4x16 OR Box_Filter_8x8) : Vertical Block number multiplied by 0 represents the offset from the V address Else if Message_seq[1] = 0 : Vertical Block number multiplied by 4 represents the offset from the V address Else if Message_seq[1] = 1 : If "Functionality = AVS" : If "Message_seq[0] = 0" : Vertical Block number multiplied by 8 represents the offset from the V address. Else Vertical Block number multiplied by 4 represents the offset from the V address. Else Vertical Block number multiplied by 1 represents the offset from the V address. Horizontal Offset: Vertical Block number multiplied by 0 represents the offset from the U address **Function =** MinMax OR BoolCentroid OR Centroid or onepixelconvolve or correlation_search Vertical Block Number is not supported and must be MBZ. Format: U9 Refer to Vertical Block Number Restriction for further details.
M1.0	31:0	**Function = AVS:** **V 2nd Derivative** Defines the change in the delta V for adjacent pixels in the Y direction. **Programming Note** **Context:** SIMD4 Payload This field is ignored for messages other than sample_8x8.

DWord	Bits	Description
		Format = IEEE_Float in normalized space For functionality other than AVS this would always be programmed to 0. **Function = Centroid:** [31:4] Reserved [3:0] Vertical Size To control the size of the centroid in vertical direction if less than 8 is required. **Function = BoolCentroid:** [31:10] Reserved [9:4] Horizontal Size To control the size of the Boolcentroid in Horizontal direction if less than 64 is required. '0' represents the default size of 64. [3:0] Vertical Size To control the size of the Boolcentroid in vertical direction if less than 4 is required. '0' represents the default size of 4. [31:16] Y-offset for Top right block from the top-left Co-ordinate [15:0] X-offset for Top right block from the top-left Co-ordinate

Restrictions for using 16x8 Message with AVS function in Sample_8x8 Message:

OFC Mode	Channels Enabled	Notes
0	All Channels	Not Supported
	Only Y-channel	Supported
	Only UV channels	Supported
1	All Channels	Not Supported
	Only Alpha Channel Disabled	Supported
	Only Y-channel	Supported
	Only UV channels	Supported
	All Other combinations	Not Supported
2	X	Supported
3	X	Supported

Vertical Block Number Restrictions

	Function	Msg Seq	Vblk Multiplier	Restrictions / Comments
Dword	M1.7	M1.7		Vertical block (vblk) is used for vertical offset except in 1D horizontal convolution where it is used as horizontal offset.
Bits	[31:28]	[26:25]		
	AVS Scaling	16x4	4	With Rotation of 90 and 270 the vblk should not be used and should be zero always.
		16x8	8	
	Convolve	16x4	4	
		16x1	1	
	MinMax		0	Vblk is not used and should be zero.
	MinMaxFilter	16x4	4	
		16x1 or 1x1	1	
	Erode or Dilate	32x1 or 64x1	1	
		32x4 or 64x4	4	
	BoolCentroid		0	Vblk is not used and should be zero.
	Centroid		0	Vblk is not used and should be zero.

Payload Parameter Definition

The table below shows all of the message types supported by the sampling engine. The **Message Type** field in the message descriptor determines which message is being sent. The **SIMD Mode** field determines the number of instances (i.e. pixels) and the formatting of the initiating and writeback messages. The **Header Present** field determines whether a header is delivered as the first phase of the message or the default header from R0 of the thread's dispatch is used. The **Message Length** field is used to vary the number of parameters sent with each message. Higher-numbered parameters are optional, and default to a value of 0 if not sent but needed for the surface being sampled. Parameter 0 is required except for the sampleinfo message for [CHV, BSW], which has no parameter 0.

The message lengths are computed as follows, where "N" is the number of parameters ("N" is rounded up to the next multiple of 4 for SIMD4x2), and "H" is 1 if the header is present, 0 otherwise. The maximum message length allowed to the sampler is 11.

SIMD Mode	Message Length
SIMD4x2	H + (N/4)
SIMD8 SIMD8D	H + N
SIMD16	H + (2*N)

The response lengths are computed as follows:

SIMD Mode		Response Length Return Format = 32-bit	Response Length Return Format = 16-bit ***
SIMD4x2		1	not allowed
SIMD8	sample+killpix	5	not allowed
	all other message types	4	2 **
SIMD16		8 *	4 *

* For SIMD16, phases in the response length are reduced by 2 for each channel that is masked.

** [CHV, BSW]: For SIMD8*, phases in the response length are reduced by 1 for each channel that is masked.

*** [CHV, BSW] only

SIMD16 messages with six or more parameters exceed the maximum message length allowed, in which case they are not supported. This includes some forms of sample_b_c, sample_l_c, and gather4_po_c message types. Note that even for these messages, if 5 or fewer parameters are included in the message, the SIMD16 form of the message is allowed. SIMD16 forms of sample_d and sample_d_c are not allowed, regardless of the number of parameters sent.

[CHV, BSW]: Response Length of zero is allowed on all SIMD8* and SIMD16* sampler messages except sample+killpix, resinfo, sampleinfo, LOD, and gather4*. Header Present must be enabled and Pixel Null Mask Enable must be disabled. The Write Channel Mask for all four channels in the header must be 0 (channel enabled). A shader containing one or more of these messages is not allowed to send any

render target read or write messages to the data port. When response length is set to zero, the following behavior occurs:

- When the sampler completes processing of the message, the resulting channels are delivered as input to the Render Cache Data Port in the form of a Render Target Write message without a header. Refer to the Data Port section for more details on this message.
- The fields normally set in the message descriptor for Render Target Write are set as follows:
  - End of Thread (EOT) comes from the EOT on the sampler message. These response length zero sampler messages are the only sampler messages allowed to have EOT enabled.
  - Last Render Target Select is set to true.
  - Slot Group Select is derived from EOT: If EOT is enabled, set to SLOTGRP_LO, otherwise SLOTGRP_HI.
  - RT Write Message Type is derived from the sampler's SIMD Mode field as follows: SIMD8 => SIMD8_LO, SIMD16 => SIMD16
  - Binding Table Index is set to the value set in the **Render Target Binding Table Index** field of the message header.

SIMD4x2 Messages

Message Type	Mnemonic	Parameters										
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
00010	sample_l	u	v	r	ai	lod						
00100	sample_d	u	v	r	ai	dudx	dudy	dvdx	dvdy	drdx	drdy	mlod
00110	sample_l_c	u	v	r	ai	ref	lod					
00111	ld	u	v	r	lod							
01000	gather4	u	v	r	ai							
01010	resinfo	lod										
01011	sampleinfo											
10000	gather4_c	u	v	r	ai	ref						
10001	gather4_po	u	v	r	ai	offu	offv					
10010	gather4_po_c	u	v	r	ref	offu	offv					
10100	sample_d_c	u	v	r	ai	dudx	dudy	dvdx	dvdy	drdx	drdy	ref
11100	ld2dms_w	u	v	r	lod	si	mcsl	mcsh				
11101	ld_mcs	u	v	r	lod							
11110	ld2dms	u	v	r	lod	si	mcs					

SIMD32/SIMD64 Messages

Message Type	mnemonic	Payload Layout	Message Length	Response Length
00000	sample_unorm	Pixel Shader	H + 1	8 **
00010	sample_unorm+killpix	Pixel Shader	H + 1	9 **
01000	deinterlace	Pixel Shader	H + 1	†
01100	sample_unorm	Media	H + 1	8 **
01010	sample_unorm+killpix	Media	H + 1	9 **
01011	sample_8x8	Media	H + 1	16 *
11111	cache_flush	no payload	1	1

SIMD32/SIMD64 Messages

Message Type	mnemonic	Payload Layout	Message Length	Response Length
00000	sample_unorm	Pixel Shader	H + 1	8 **
00010	sample_unorm+killpix	Pixel Shader	H + 1	9 **
01000	deinterlace	Pixel Shader	H + 1	†
01100	sample_unorm	Media	H + 1	8 **
01010	sample_unorm+killpix	Media	H + 1	9 **
01011	sample_8x8	Media	H + 1	16 *
11111	cache_flush	no payload	1	1

* For sample_8x8, phases in the response length are reduced by 4 for each channel that is masked.

** For sample_unorm, phases in the response length are reduced by 2 for each channel that is masked.

† For deinterlace, response length depending on certain state fields. Refer to writeback message definition for details.

SIMD32_64 Message Descriptor

Please refer to the 3D Sampler Message Descriptor definition at **Message Descriptor - Sampling Engine**.

SIMD32_64 Message Header

Please refer to the 3D Sampler Message Header definition at Message Header.

Message Header

The message header for the sampling engine is the same regardless of the message type. If the header is not present, the behavior is as if the message was sent with all fields in the header set to zero (write channel masks are all enabled and offsets are zero). When Response length is 0 for sample_8x8 message then the data from sampler is directly written out to memory using media write message. Message header needs to be present if mid-thread pre-emption is required.

DWord	Bits	Description
M0.5	31:0	Ignored
M0.4	31:0	Reserved
M0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the sampler state table. This field is ignored for “ld” and “resinfo” message types. This pointer is relative to the **Dynamic State Base Address**. Format = DynamicStateOfffset[31:5]
	4:1	Ignored
	0	Ignored
M0.2	31:24	Ignored
	31:24	**Render Target or Destination Binding Table Index:** Specifies the index into the binding table for the render target or HDC for messages with response length of zero (the binding table index for the sampler surface is in the message descriptor). Format = U8 Range = [0,255]
	23:22	Reserved
	21	Ignored
	20	Ignored
	19:18	**SIMD32/64 Output Format Control** Specifies the output format of SIMD32/64 messages (sample_unorm* and sample_8x8). Ignored for other message types. Refer to the writeback message formats for details on how this field affects returned data. 0: 16 bit Full 1: 16 bit Chrominance Downsampled 2: 8 bit Full 3: 8 bit Chrominance Downsampled **Programming Note** **Context:** Message Header This field is ignored for sample_8x8 messages if the Function is not AVS and MinMaxFilter. For

DWord	Bits	Description
		MinMaxFilter only 16bit Full and 8bit Full modes are supported.
	17:16	**Gather4 Source Channel Select:** Selects the source channel to be sampled in the gather4* messages. Ignored for other message types. 0: Red channel 1: Green channel 2: Blue channel 3: Alpha channel Note that for gather4*_c messages, this field must be set to 0 (Red channel).
	15	**Alpha Write Channel Mask:** Enables the alpha channel to be written back to the originating thread. 0: Alpha channel is written back. 1: Alpha channel is not written back. **Programming Note** **Context:** Message Header • A message with all four channels masked is not allowed. • This field is ignored for the deinterlace message. • This field must be set to zero for sample_8x8 in VSA mode. • This field must be set to zero for all gather4* messages. **Programming Note** **Context:** Message Header For Sample_8x8 messages, Alpha/Blue/Red channels should be always masked (set to 1) and only Green channel is enabled (set to 0).
	14	**Blue Write Channel Mask:** See Alpha Write Channel Mask.
	13	**Green Write Channel Mask:** See Alpha Write Channel Mask.
	12	**Red Write Channel Mask:** See Alpha Write Channel Mask.
	11:8	**U Offset:** The u offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note** **Context:** Message Header

DWord	Bits	Description
		• This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages. • This field is ignored if the "offu" parameter is included in the gather4* messages.
	7:4	**V Offset:** The v offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. **Programming Note** **Context:** Message Header • This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages. • This field is ignored if the "offu" parameter is included in the gather4* messages.
	3:0	**R Offset:** The r offset from the _aoffimmi modifier on the "sample" or "ld" instruction in DX10. Must be zero if the **Surface Type** is SURFTYPE_CUBE or SURFTYPE_BUFFER. Must be set to zero if _aoffimmi is not specified. Format is S3 2's complement. This field is ignored for the sample_unorm*, sample_8x8, and deinterlace messages.
M0.1	31:0	Ignored
M0.0	31:0	Ignored

SIMD32_64 Payload Parameter Definition

Please refer to the 3D Sampler Payload Parameter Definition at **Payload Parameter Definition**.

SIMD32_64 Message Types

Please refer to the 3D Sampler Message Types definition at **Message Types**.

Writeback Message

SIMD32

Sample_unorm*

Pixels are numbered as follows:

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31

Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in both destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel. For example, if only red and alpha are enabled, red is sent to regid+0 and regid+1, and alpha to regid+2 and regid+3 (using 16 bit Full mode as an example).

"16 bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 11 & 10 Red**
W0.4		**Pixel 9 & 8 Red**
W0.3		**Pixel 7 & 6 Red**
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 31 & 30 Red**
W1.6		**Pixel 29 & 28 Red**
W1.5		**Pixel 27 & 26 Red**
W1.4		**Pixel 25 & 24 Red**
W1.3		**Pixel 23 & 22 Red**
W1.2		**Pixel 21 & 20 Red**
W1.1		**Pixel 19 & 18 Red**
W1.0		**Pixel 17 & 16 Red**
W2		**Pixels 15:0 Green**

DWord	Bit	Description
W3		**Pixels 31:16 Green**
W4		**Pixels 15:0 Blue**
W5		**Pixels 31:16 Blue**
W6		**Pixels 15:0 Alpha**
W7		**Pixels 31:16 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bit	Description
W0.7	31:16	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 28 Red**
W0.6		**Pixel 26 & 24 Red**
W0.5		**Pixel 22 & 20 Red**
W0.4		**Pixel 18 & 16 Red**
W0.3		**Pixel 14 & 12 Red**
W0.2		**Pixel 10 & 8 Red**
W0.1		**Pixel 6 & 4 Red**
W0.0		**Pixel 2 & 0 Red**
W1.7	31:16	**Pixel 15 Green**
	15:0	**Pixel 14 Green**
W1.6		**Pixel 13 & 12 Green**
W1.5		**Pixel 11 & 10 Green**
W1.4		**Pixel 9 & 8 Green**
W1.3		**Pixel 7 & 6 Green**
W1.2		**Pixel 5 & 4 Green**
W1.1		**Pixel 3 & 2 Green**
W1.0		**Pixel 1 & 0 Green**
W2.7		**Pixel 31 & 30 Green**
W2.6		**Pixel 29 & 28 Green**
W2.5		**Pixel 27 & 26 Green**
W2.4		**Pixel 25 & 24 Green**
W2.3		**Pixel 23 & 22 Green**
W2.2		**Pixel 21 & 20 Green**
W2.1		**Pixel 19 & 18 Green**

DWord	Bit	Description
W2.0		**Pixel 17 & 16 Green**
W3.7	31:16	**Pixel 30 Blue**
	15:0	**Pixel 28 Blue**
W3.6		**Pixel 26 & 24 Blue**
W3.5		**Pixel 22 & 20 Blue**
W3.4		**Pixel 18 & 16 Blue**
W3.3		**Pixel 14 & 12 Blue**
W3.2		**Pixel 10 & 8 Blue**
W3.1		**Pixel 6 & 4 Blue**
W3.0		**Pixel 2 & 0 Blue**
W4.7	31:16	**Pixel 15 Alpha**
	15:0	**Pixel 14 Alpha**
W4.6		**Pixel 13 & 12 Alpha**
W4.5		**Pixel 11 & 10 Alpha**
W4.4		**Pixel 9 & 8 Alpha**
W4.3		**Pixel 7 & 6 Alpha**
W4.2		**Pixel 5 & 4 Alpha**
W4.1		**Pixel 3 & 2 Alpha**
W4.0		**Pixel 1 & 0 Alpha**
W5.7		**Pixel 31 & 30 Alpha**
W5.6		**Pixel 29 & 28 Alpha**
W5.5		**Pixel 27 & 26 Alpha**
W5.4		**Pixel 25 & 24 Alpha**
W5.3		**Pixel 23 & 22 Alpha**
W5.2		**Pixel 21 & 20 Alpha**
W5.1		**Pixel 19 & 18 Alpha**
W5.0		**Pixel 17 & 16 Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:24	**Pixel 31 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 30 Red**
	15:8	**Pixel 29 Red**
	7:0	**Pixel 28 Red**

DWord	Bit	Description
W0.6		**Pixel 27:24 Red**
W0.5		**Pixel 23:20 Red**
W0.4		**Pixel 19:16 Red**
W0.3		**Pixel 15:12 Red**
W0.2		**Pixel 11:8 Red**
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1		**Pixels 31:0 Green**
W2		**Pixels 31:0 Blue**
W3		**Pixels 31:0 Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

If either red or blue channel (but not both) are masked, the W0 register is included in the payload but the masked channel is not written to the GRF. If both are masked, W0 is not included in the payload (reducing the response length by one).

DWord	Bit	Description
W0.7	31:24	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	23:16	**Pixel 28 Red**
	15:8	**Pixel 26 Red**
	7:0	**Pixel 24 Red**
W0.6		**Pixel 22, 20, 18, 16 Red**
W0.5		**Pixel 14, 12, 10, 8 Red**
W0.4		**Pixel 6, 4, 2, 0 Red**
W0.3		**Pixel 30, 28, 26, 24 Blue**
W0.2		**Pixel 22, 20, 18, 16 Blue**
W0.1		**Pixel 14, 12, 10, 8 Blue**
W0.0		**Pixel 6, 4, 2, 0 Blue**
W1.7	31:24	**Pixel 31 Green**
	23:16	**Pixel 30 Green**
	15:8	**Pixel 29 Green**
	7:0	**Pixel 28 Green**
W1.6		**Pixel 27:24 Green**
W1.5		**Pixel 23:20 Green**
W1.4		**Pixel 19:16 Green**

DWord	Bit	Description
W1.3		**Pixel 15:12 Green**
W1.2		**Pixel 11:8 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2.7		**Pixel 31:28 Alpha**
W2.6		**Pixel 27:24 Alpha**
W2.5		**Pixel 23:20 Alpha**
W2.4		**Pixel 19:16 Alpha**
W2.3		**Pixel 15:12 Alpha**
W2.2		**Pixel 11:8 Alpha**
W2.1		**Pixel 7:4 Alpha**
W2.0		**Pixel 3:0 Alpha**

Additional Writeback Phase for sample_unorm+killpix

For the sample_unorm+killpix messages, an additional writeback phase is returned. The value of "n" depends on which channels are enabled for return and the **Output Format Control Mode**, this register will immediately follow the first part of the writeback message.

DWord	Bit	Description
Wn.7:1		Reserved (not written)
Wn.0	31:0	**Active Pixel Mask:** This field has the bit for all pixels set to 1 except those pixels that have been killed as a result of chroma key with kill pixel mode. The bits in this mask correspond to the pixels as follows and they are listed from upper left (MSB) lower right LSB: 31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Cache_flush

The writeback message is for cache_flush indicates that the flush has been completed. The destination register is not modified.

DWord	Bit	Description
W0.7:0		Reserved

Sample_8x8 Writeback Messages

The writeback message for sample_8x8 consists of up to 16 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in all four destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel.

Pixels are numbered as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

"16 bit Full" Output Format Control Mode

DWord	Bits	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 11 & 10 Red**
W0.4		**Pixel 9 & 8 Red**
W0.3		**Pixel 7 & 6 Red**
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1		**Pixel 31:16 Red**
W2		**Pixels 47:32 Red**
W3		**Pixels 63:33 Red**
W4		**Pixels 15:0 Green**
W5		**Pixels 31:16 Green**
W6		**Pixels 47:32 Green**
W7		**Pixels 63:33 Green**
W8		**Pixels 15:0 Blue**
W9		**Pixels 31:16 Blue**
W10		**Pixels 47:32 Blue**
W11		**Pixels 63:33 Blue**
W12		**Pixels 15:0 Alpha**
W13		**Pixels 31:16 Alpha**
W14		**Pixels 47:32 Alpha**
W15		**Pixels 63:33 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bits	Description
W0.7	31:16	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 28 Red**
W0.6		**Pixel 26 & 24 Red**
W0.5		**Pixel 22 & 20 Red**
W0.4		**Pixel 18 & 16 Red**
W0.3		**Pixel 14 & 12 Red**
W0.2		**Pixel 10 & 8 Red**
W0.1		**Pixel 6 & 4 Red**
W0.0		**Pixel 2 & 0 Red**
W1.7		**Pixel 62 & 60 Red**
W1.6		**Pixel 58 & 56 Red**
W1.5		**Pixel 54 & 52 Red**
W1.4		**Pixel 50 & 48 Red**
W1.3		**Pixel 46 & 44 Red**
W1.2		**Pixel 42 & 40 Red**
W1.1		**Pixel 38 & 36 Red**
W1.0		**Pixel 34 & 32 Red**
W2.7	31:16	**Pixel 15 Green**
	15:0	**Pixel 14 Green**
W2.6		**Pixel 13 & 12 Green**
W2.5		**Pixel 11 & 10 Green**
W2.4		**Pixel 9 & 8 Green**
W2.3		**Pixel 7 & 6 Green**
W2.2		**Pixel 5 & 4 Green**
W2.1		**Pixel 3 & 2 Green**
W2.0		**Pixel 1 & 0 Green**
W3		**Pixel 31:16 Green**
W4		**Pixel 47:32 Green**
W5		**Pixel 63:48 Green**
W6.7	31:16	**Pixel 30 Blue**
	15:0	**Pixel 28 Blue**
W6.6		**Pixel 26 & 24 Blue**
W6.5		**Pixel 22 & 20 Blue**

DWord	Bits	Description
W6.4		**Pixel 18 & 16 Blue**
W6.3		**Pixel 14 & 12 Blue**
W6.2		**Pixel 10 & 8 Blue**
W6.1		**Pixel 6 & 4 Blue**
W6.0		**Pixel 2 & 0 Blue**
W7.7		**Pixel 62 & 60 Blue**
W7.6		**Pixel 58 & 56 Blue**
W7.5		**Pixel 54 & 52 Blue**
W7.4		**Pixel 50 & 48 Blue**
W7.3		**Pixel 46 & 44 Blue**
W7.2		**Pixel 42 & 40 Blue**
W7.1		**Pixel 38 & 36 Blue**
W7.0		**Pixel 34 & 32 Blue**
W8.7	31:16	**Pixel 15 Alpha**
	15:0	**Pixel 14 Alpha**
W8.6		**Pixel 13 & 12 Alpha**
W8.5		**Pixel 11 & 10 Alpha**
W8.4		**Pixel 9 & 8 Alpha**
W8.3		**Pixel 7 & 6 Alpha**
W8.2		**Pixel 5 & 4 Alpha**
W8.1		**Pixel 3 & 2 Alpha**
W8.0		**Pixel 1 & 0 Alpha**
W9		**Pixel 31:16 Alpha**
W10		**Pixel 47:32 Alpha**
W11		**Pixel 63:48 Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 31 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 30 Red**
	15:8	**Pixel 29 Red**
	7:0	**Pixel 28 Red**
W0.6		**Pixel 27:24 Red**
W0.5		**Pixel 23:20 Red**

DWord	Bits	Description
W0.4		**Pixel 19:16 Red**
W0.3		**Pixel 15:12 Red**
W0.2		**Pixel 11:8 Red**
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1.7		**Pixel 63:60 Red**
W1.6		**Pixel 59:56 Red**
W1.5		**Pixel 55:52 Red**
W1.4		**Pixel 51:48 Red**
W1.3		**Pixel 47:44 Red**
W1.2		**Pixel 43:40 Red**
W1.1		**Pixel 39:36 Red**
W1.0		**Pixel 35:52 Red**
W2		**Pixels 31:0 Green**
W3		**Pixels 63:32 Green**
W4		**Pixels 31:0 Blue**
W5		**Pixels 63:32 Blue**
W6		**Pixels 31:0 Alpha**
W7		**Pixels 63:32 Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 62 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 60 Red**
	15:8	**Pixel 58 Red**
	7:0	**Pixel 56 Red**
W0.6		**Pixel 54, 52, 50, 48 Red**
W0.5		**Pixel 46, 44, 42, 40 Red**
W0.4		**Pixel 38, 36, 34, 32 Red**
W0.3		**Pixel 30, 28, 26, 24 Red**
W0.2		**Pixel 22, 20, 18, 16 Red**
W0.1		**Pixel 14, 12, 10, 8 Red**
W0.0		**Pixel 6, 4, 2, 0 Red**
W1.7	31:24	**Pixel 31 Green**

DWord	Bits	Description
	23:16	**Pixel 30 Green**
	15:8	**Pixel 29 Green**
	7:0	**Pixel 28 Green**
W1.6		**Pixel 27:24 Green**
W1.5		**Pixel 23:20 Green**
W1.4		**Pixel 19:16 Green**
W1.3		**Pixel 15:12 Green**
W1.2		**Pixel 11:8 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2		**Pixel 63:32 Green**
W3.7	31:24	**Pixel 62 Blue**
	23:16	**Pixel 60 Blue**
	15:8	**Pixel 58 Blue**
	7:0	**Pixel 56 Blue**
W3.6		**Pixel 54, 52, 50, 48 Blue**
W3.5		**Pixel 46, 44, 42, 40 Blue**
W3.4		**Pixel 38, 36, 34, 32 Blue**
W3.3		**Pixel 30, 28, 26, 24 Blue**
W3.2		**Pixel 22, 20, 18, 16 Blue**
W3.1		**Pixel 14, 12, 10, 8 Blue**
W3.0		**Pixel 6, 4, 2, 0 Blue**
W4.7	31:24	**Pixel 31 Alpha**
	23:16	**Pixel 30 Alpha**
	15:8	**Pixel 29 Alpha**
	7:0	**Pixel 28 Alpha**
W4.6		**Pixel 27:24 Alpha**
W4.5		**Pixel 23:20 Alpha**
W4.4		**Pixel 19:16 Alpha**
W4.3		**Pixel 15:12 Alpha**
W4.2		**Pixel 11:8 Alpha**
W4.1		**Pixel 7:4 Alpha**
W4.0		**Pixel 3:0 Alpha**
W5		**Pixel 63:32 Alpha**

Sampler_8x8 – Writeback Message for Convolve and 1 Pixel Convolution and 1D Vertical Convolve

DWord	Bits	Description
W0.7		**Pixel 15 & 14 Row 0**
W0.6		**Pixel 13 & 12 Row 0**
W0.5		**Pixel 11 & 10 Row 0**
W0.4		**Pixel 9 & 8 Row 0**
W0.3		**Pixel 7 & 6 Row 0**
W0.2		**Pixel 5 & 4 Row 0**
W0.1		**Pixel 3 & 2 Row 0**
W0.0	31:16	**Pixel 1 Row 0** Format = 16-bit SINT
	15:0	**Pixel 0 Row 0**
W1		**Pixel [15:0] Row 1** Only for 16x4 Message. Not present for 16x1 and 1x1 message.
W2		**Pixel [15:0] Row 2** Only for 16x4 Message. Not present for 16x1 and 1x1 message.
W3		**Pixel [15:0] Row 3** Only for 16x4 Message. Not present for 16x1 and 1x1 message.

Sampler_8x8 – Writeback Message for MinMaxFilter

"16 Bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:24	**Pixel 7 Max MSB Row 0 if input is 16bits** **Pixel 7 Max Row 0 if input is 8bits** **Reserved if 1x1 message**
	23:16	**Pixel 7 Max LSB Row 0 if input is 16bits** **Reserved if 1x1 message (or) 8bit input**
	15:8	**Pixel 7 Min MSB Row 0 if input is 16bits** **Pixel 7 Min Row 0 if input is 8bits** **Reserved if 1x1 message**
	7:0	**Pixel 7 Min LSB Row 0 if input is 16bits** **Reserved if 1x1 message (or) 8bit input**
W0.6		**Pixel 6 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)** **Reserved if 1x1 message**
W0.5		**Pixel 5 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)** **Reserved if 1x1 message**
W0.4		**Pixel 4 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)** **Reserved if 1x1 message**
W0.3		**Pixel 3 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)** **Reserved if 1x1 message**
W0.2		**Pixel 2 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)**

DWord	Bit	Description
		Reserved if 1x1 message
W0.1		**Pixel 1 Min/Max Row 0** **(Format same as above depending on 8bit or 16bit)** **Reserved if 1x1 message**
W0.0		**Pixel 0 Row 0 if input is 16bits** **[31:0] Pixel 0 Max** **[15:0] Pixel 0 Min** **Pixel 0 Row 0 if input is 8bits** **[31:24] Pixel 0 Max** **[23:16] Reserved** **[15:8] Pixel 0 Min** **[7:0] Reserved** **The Min or Max could be disabled. Then the corresponding values is invalid and should be ignored.**
W1		**Pixel [15:8] Row 0**
W2		**Pixel [7:0] Row 1** **Only for 16x4 Message. Not present for 16x1 message**
W3		**Pixel [15:8] Row 1** **Only for 16x4 Message. Not present for 16x1 message**
W4		**Pixel [7:0] Row 2** **Only for 16x4 Message. Not present for 16x1 message**
W5		**Pixel [15:8] Row 2** **Only for 16x4 Message. Not present for 16x1 message**
W6		**Pixel [7:0] Row 3** **Only for 16x4 Message. Not present for 16x1 message**
W7		**Pixel [15:8] Row 3** **Only for 16x4 Message. Not present for 16x1 message**

"8 Bit Full" Output Format Control Mode:

DWord	Bit	Description
W0.7	31:24	**Pixel 15 Max Row 0** **Reserved if 1x1 message**
	23:16	**Pixel 14 Max Row 0** **Reserved if 1x1 message**
	15:8	**Pixel 15 Min Row 0** **Reserved if 1x1 message**
	7:0	**Pixel 14 Min Row 0** **Reserved if 1x1 message**
W0.6		**Pixel [13:12] Min/Max Row 0** **Reserved if 1x1 message**
W0.5		**Pixel [11:10] Min/Max Row 0** **Reserved if 1x1 message**
W0.4		**Pixel [9:8] Min/Max Row 0** **Reserved if 1x1 message**
W0.3		**Pixel [7:6] Min/Max Row 0** **Reserved if 1x1 message**
W0.2		**Pixel [5:4] Min/Max Row 0** **Reserved if 1x1 message**
W0.1		**Pixel [3:2] Min/Max Row 0** **Reserved if 1x1 message**
W0.0		**Pixel [1:0] Min/Max Row 0** **Reserved if 1x1 message**
W1		**Pixel [15:0] Row 1** **Only for 16x4 Message. Not present for 16x1 message**

DWord	Bit	Description
W2		**Pixel [15:0] Row 2** **Only for 16x4 Message. Not present for 16x1 message**
W3		**Pixel [15:0] Row 3** **Only for 16x4 Message. Not present for 16x1 message**

Sampler_8x8 – Writeback Message for MinMax

DWord	Bit	Description
W0.7	31:0	**Reserved**
W0.6	31:0	**Reserved**
W0.5	31:0	**Reserved**
W0.4	31:0	**Reserved**
W0.3	31:0	**Reserved**
W0.2	31:0	**Reserved**
W0.1	31:0	**Reserved**
W0.0	31:24	**Pixel 0 Max MSB if input is 16bits** **Pixel 0 Max Row 0 if input is 8bits**
	23:16	**Pixel 0 Max LSB if input is 16bits** **Reserved if input is 8bits**
	15:8	**Pixel 0 Min MSB if input is 16bits** **Pixel 0 Min Row 0 if input is 8bits**
	7:0	**Pixel 0 Min LSB if input is 16bits** **Reserved if input is 8bits**

Sampler_8x8 – Writeback Message for Dilate or Erode

DWord	Bit	Description
W0.7	31:0	**Pixel [63:32] Row 3** **Only for 64x4 message. Not present for 32x4 or 64x1 or 32x1 message**
W0.6		**Pixel [31:0] Row 3** **Only for 64x4 or 32x4 Message. Not present for 64x1 or 32x1 message**
W0.5		**Pixel [63:32] Row 2** **Only for 64x4 message. Not present for 32x4 or 64x1 or 32x1 message**
W0.4		**Pixel [31:0] Row 2** **Only for 64x4 or 32x4 Message. Not present for 64x1 or 32x1 message**
W0.3		**Pixel [63:32] Row 1** **Only for 64x4 message. Not present for 32x4 or 64x1 or 32x1 message**
W0.2		**Pixel [31:0] Row 1** **Only for 64x4 or 32x4 Message. Not present for 64x1 or 32x1 message**
W0.1		**Pixel [63:32] Row 0** **Only for 64x4 or 64x1 Message. Not present for 32x4 or 32x1 message**
W0.0		**Pixel [31:0] Row 0**

Sampler_8x8 – Writeback Message for Centroid

DWord	Bit	Description
W0.7	31:0	**Divisor/Sum Column 3**
W0.6		**jSum Column 3**
W0.5		**Divisor/Sum Column 2**
W0.4		**jSum Column 2**
W0.3		**Divisor/Sum Column 1**
W0.2		**jSum Column 1**
W0.1		**Divisor/Sum Column 0**
W0.0		**jSum Column 0**
W1		**Divisor/Sum & jSum for Column [7:4]**
W2		**Divisor/Sum & jSum for Column [11:8]**
W3		**Divisor/Sum & jSum for Column [15:12]**

Sampler_8x8 – Writeback Message for BoolCentroid/BoolSum

DWord	Bit	Description
W0.7	31:0	**Reserved**
W0.6	31:0	**Sum 16b Column 1**
W0.5	31:0	**jSum 16b Column 1**
W0.4	31:0	**iSum 16b Column 1**
W0.3	31:0	**Reserved**
W0.2	31:0	**Sum 16b Column 0**
W0.1	31:0	**jSum 16b Column 0**
W0.0	31:0	**iSum 16b Column 0**
W1		**Column[3:2] same as above**

Sampler_8x8 – Writeback Message for AVS

The writeback message for sample_8x8 consists of up to 16 destination registers for 16x4 and 16x8 message type. 16x8 AVS messages have restrictions mentioned in section SIMD32/64 payload for [CHV, BSW] which limits the writeback to only 16 destination registers. Which registers are returned is determined by the write channel mask received in the corresponding input message. Each asserted write channel mask results in all four destination registers of the corresponding channel being skipped in the writeback message, and all channels with higher numbered registers being dropped down to fill in the space occupied by the masked channel.

Pixels are numbered as follows:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63

"16 bit Full" Output Format Control Mode

DWord	Bit	Description
W0.7	31:16	**Pixel 15 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 14 Red**
W0.6		**Pixel 13 & 12 Red**
W0.5		**Pixel 11 & 10 Red**
W0.4		**Pixel 9 & 8 Red**
W0.3		**Pixel 7 & 6 Red**

DWord	Bit	Description
W0.2		**Pixel 5 & 4 Red**
W0.1		**Pixel 3 & 2 Red**
W0.0		**Pixel 1 & 0 Red**
W1.7		**Pixel 31 & 30 Red**
W1.6		**Pixel 29 & 28 Red**
W1.5		**Pixel 27 & 26 Red**
W1.4		**Pixel 25 & 24 Red**
W1.3		**Pixel 23 & 22 Red**
W1.2		**Pixel 21 & 20 Red**
W1.1		**Pixel 19 & 18 Red**
W1.0		**Pixel 17 & 16 Red**
W2		**Pixels 47:32 Red**
W3		**Pixels 63:48 Red**
W4		**Pixels 15:0 Green**
W5		**Pixels 31:16 Green**
W6		**Pixels 47:32 Green**
W7		**Pixels 63:48 Green**
W8		**Pixels 15:0 Blue**
W9		**Pixels 31:16 Blue**
W10		**Pixels 47:32 Blue**
W11		**Pixels 63:48 Blue**
W12		**Pixels 15:0 Alpha**
W13		**Pixels 31:16 Alpha**
W14		**Pixels 47:32 Alpha**
W15		**Pixels 63:48 Alpha**
		Below is valid only for 16x8 messages
W16		**Pixels 79:64 Red**
W17		**Pixels 95:80 Red**
W18		**Pixels 111:96 Red**
W19		**Pixels 127:112 Red**
W20		**Pixels 79:64 Green**
W21		**Pixels 95:80 Green**
W22		**Pixels 111:96 Green**
W23		**Pixels 127:112 Green**
W24		**Pixels 79:64 Blue**
W25		**Pixels 95:80 Blue**

DWord	Bit	Description
W26		**Pixels 111:96 Blue**
W27		**Pixels 127:112 Blue**
W28		**Pixels 111:96 Alpha**
W29		**Pixels 127:112 Alpha**
W30		**Pixels 111:96 Alpha**
W31		**Pixels 127:112 Alpha**

"16 Bit Chrominance Downsampled" Output Format Control Mode

In this mode the odd pixel red & blue channels are not included.

DWord	Bits	Description
W0.7	31:16	**Pixel 30 Red** Format = 16-bit UNORM with an 8-bit range (the value FF00h maps to a real value of 1.0) Range = [0000h:FF00h]
	15:0	**Pixel 28 Red**
W0.6		**Pixel 26 & 24 Red**
W0.5		**Pixel 22 & 20 Red**
W0.4		**Pixel 18 & 16 Red**
W0.3		**Pixel 14 & 12 Red**
W0.2		**Pixel 10 & 8 Red**
W0.1		**Pixel 6 & 4 Red**
W0.0		**Pixel 2 & 0 Red**
W1.7		**Pixel 62 & 60 Red**
W1.6		**Pixel 58 & 56 Red**
W1.5		**Pixel 54 & 52 Red**
W1.4		**Pixel 50 & 48 Red**
W1.3		**Pixel 46 & 44 Red**
W1.2		**Pixel 42 & 40 Red**
W1.1		**Pixel 38 & 36 Red**
W1.0		**Pixel 34 & 32 Red**
W2.7	31:16	**Pixel 15 Green**
	15:0	**Pixel 14 Green**
W2.6		**Pixel 13 & 12 Green**
W2.5		**Pixel 11 & 10 Green**
W2.4		**Pixel 9 & 8 Green**
W2.3		**Pixel 7 & 6 Green**

DWord	Bits	Description
W2.2		**Pixel 5 & 4 Green**
W2.1		**Pixel 3 & 2 Green**
W2.0		**Pixel 1 & 0 Green**
W3		**Pixel 31:16 Green**
W4		**Pixel 47:32 Green**
W5		**Pixel 63:48 Green**
W6.7	31:16	**Pixel 30 Blue**
	15:0	**Pixel 28 Blue**
W6.6		**Pixel 26 & 24 Blue**
W6.5		**Pixel 22 & 20 Blue**
W6.4		**Pixel 18 & 16 Blue**
W6.3		**Pixel 14 & 12 Blue**
W6.2		**Pixel 10 & 8 Blue**
W6.1		**Pixel 6 & 4 Blue**
W6.0		**Pixel 2 & 0 Blue**
W7.7		**Pixel 62 & 60 Blue**
W7.6		**Pixel 58 & 56 Blue**
W7.5		**Pixel 54 & 52 Blue**
W7.4		**Pixel 50 & 48 Blue**
W7.3		**Pixel 46 & 44 Blue**
W7.2		**Pixel 42 & 40 Blue**
W7.1		**Pixel 38 & 36 Blue**
W7.0		**Pixel 34 & 32 Blue**
W8.7	31:16	**Pixel 15 Alpha**
	15:0	**Pixel 14 Alpha**
W8.6		**Pixel 13 & 12 Alpha**
W8.5		**Pixel 11 & 10 Alpha**
W8.4		**Pixel 9 & 8 Alpha**
W8.3		**Pixel 7 & 6 Alpha**
W8.2		**Pixel 5 & 4 Alpha**
W8.1		**Pixel 3 & 2 Alpha**
W8.0		**Pixel 1 & 0 Alpha**
W9		**Pixel 31:16 Alpha**
W10		**Pixel 47:32 Alpha**
W11		**Pixel 63:48 Alpha**
		Below DWs is only valid for 16x8 message

DWord	Bits	Description
W12		**Pixel [94,92,90,88,86,84,82,80,78,76,74,72,70,68,66,64] Red**
W13		**Pixel [126,124,122,120,118,116,114,112,110,108,106,104,102,100,98,96] Red**
W14		**Pixel [79:64] Green**
W15		**Pixel [95:80] Green**
W16		**Pixel [111:96] Green**
W17		**Pixel [127:112] Green**
W18		**Pixel [94,92,90,88,86,84,82,80,78,76,74,72,70,68,66,64] Blue**
W19		**Pixel [126,124,122,120,118,116,114,112,110,108,106,104,102,100,98,96] Blue**
W20		**Pixel [79:64] Alpha**
W21		**Pixel [95:80] Alpha**
W22		**Pixel [111:96] Alpha**
W23		**Pixel [127:112] Alpha**

"8 Bit Full" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 31 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 30 Red**
	15:8	**Pixel 29 Red**
	7:0	**Pixel 28 Red**
W0.6		**Pixel 27:24 Red**
W0.5		**Pixel 23:20 Red**
W0.4		**Pixel 19:16 Red**
W0.3		**Pixel 15:12 Red**
W0.2		**Pixel 11:8 Red**
W0.1		**Pixel 7:4 Red**
W0.0		**Pixel 3:0 Red**
W1.7		**Pixel 63:60 Red**
W1.6		**Pixel 59:56 Red**
W1.5		**Pixel 55:52 Red**
W1.4		**Pixel 51:48 Red**
W1.3		**Pixel 47:44 Red**
W1.2		**Pixel 43:40 Red**
W1.1		**Pixel 39:36 Red**

DWord	Bits	Description
W1.0		**Pixel 35:52 Red**
W2		**Pixels 31:0 Green**
W3		**Pixels 63:32 Green**
W4		**Pixels 31:0 Blue**
W5		**Pixels 63:32 Blue**
W6		**Pixels 31:0 Alpha**
W7		**Pixels 63:32 Alpha**
		Below DWs is only valid for 16x8 message
W8		**Pixel [95:64] Red**
W9		**Pixel [127:96] Red**
W10		**Pixel [95:64] Green**
W11		**Pixel [127:96] Green**
W12		**Pixel [95:64] Blue**
W13		**Pixel [127:96] Blue**
W14		**Pixel [95:64] Alpha**
W15		**Pixel [127:96] Alpha**

"8 Bit Chrominance Downsampled" Output Format Control Mode

DWord	Bits	Description
W0.7	31:24	**Pixel 62 Red** Format = 8-bit UNORM Range = [00h:FFh]
	23:16	**Pixel 60 Red**
	15:8	**Pixel 58 Red**
	7:0	**Pixel 56 Red**
W0.6		**Pixel 54, 52, 50, 48 Red**
W0.5		**Pixel 46, 44, 42, 40 Red**
W0.4		**Pixel 38, 36, 34, 32 Red**
W0.3		**Pixel 30, 28, 26, 24 Red**
W0.2		**Pixel 22, 20, 18, 16 Red**
W0.1		**Pixel 14, 12, 10, 8 Red**
W0.0		**Pixel 6, 4, 2, 0 Red**
W1.7	31:24	**Pixel 31 Green**
	23:16	**Pixel 30 Green**
	15:8	**Pixel 29 Green**

DWord	Bits	Description
	7:0	**Pixel 28 Green**
W1.6		**Pixel 27:24 Green**
W1.5		**Pixel 23:20 Green**
W1.4		**Pixel 19:16 Green**
W1.3		**Pixel 15:12 Green**
W1.2		**Pixel 11:8 Green**
W1.1		**Pixel 7:4 Green**
W1.0		**Pixel 3:0 Green**
W2		**Pixel 63:32 Green**
W3.7	31:24	**Pixel 62 Blue**
	23:16	**Pixel 60 Blue**
	15:8	**Pixel 58 Blue**
	7:0	**Pixel 56 Blue**
W3.6		**Pixel 54, 52, 50, 48 Blue**
W3.5		**Pixel 46, 44, 42, 40 Blue**
W3.4		**Pixel 38, 36, 34, 32 Blue**
W3.3		**Pixel 30, 28, 26, 24 Blue**
W3.2		**Pixel 22, 20, 18, 16 Blue**
W3.1		**Pixel 14, 12, 10, 8 Blue**
W3.0		**Pixel 6, 4, 2, 0 Blue**
W4.7	31:24	**Pixel 31 Alpha**
	23:16	**Pixel 30 Alpha**
	15:8	**Pixel 29 Alpha**
	7:0	**Pixel 28 Alpha**
W4.6		**Pixel 27:24 Alpha**
W4.5		**Pixel 23:20 Alpha**
W4.4		**Pixel 19:16 Alpha**
W4.3		**Pixel 15:12 Alpha**
W4.2		**Pixel 11:8 Alpha**
W4.1		**Pixel 7:4 Alpha**
W4.0		**Pixel 3:0 Alpha**
W5		**Pixel 63:32 Alpha**
		Below DWs is only valid for 16x8 message
W6.7: 6.4		**Pixel [126,124,122,120,118,116,114,112,110,108,106,104,102,100,98,96] Red**
W6.3: 6.0		**Pixel [94,92,90,88,86,84,82,80,78,76,74,72,70,68,66,64] Red**
W7		**Pixel [95:64] Green**

DWord	Bits	Description
W8		**Pixel [127:96] Green**
W9.7: 9.4		**Pixel [126,124,122,120,118,116,114,112,110,108,106,104,102,100,98,96] Blue**
W9.3 :9.0		**Pixel [94,92,90,88,86,84,82,80,78,76,74,72,70,68,66,64] Blue**
W10		**Pixel [95:64] Alpha**
W17		**Pixel [127:96] Alpha**

SIMD32 Surface State

Please refer to the 3D Surface State definition in the **SURFACE_STATE** topic.

SIMD32 Sampler State

Please refer to the 3D Sampler State definition at *Sampler State.*

3D Pipeline Stages

The following table lists the various stages of the 3D pipeline and describes their major functions.

Pipeline Stage	Functions Performed
Command Stream (CS)	The Command Stream stage is responsible for managing the 3D pipeline and passing commands down the pipeline. In addition, the CS unit reads "constant data" from memory buffers and places it in the URB. Note that the CS stage is shared between the 3D, GPGPU and Media pipelines.
Vertex Fetch (VF)	The Vertex Fetch stage, in response to 3D Primitive Processing commands, is responsible for reading vertex data from memory, reformatting it, and writing the results into Vertex URB Entries. It then outputs primitives by passing references to the VUEs down the pipeline.
Vertex Shader (VS)	The Vertex Shader stage is responsible for processing (shading) incoming vertices by passing them to VS threads.
Hull Shader (HS)	The Hull Shader is responsible for processing (shading) incoming patch primitives as part of the tessellation process.
Tessellation Engine (TE)	The Tessellation Engine is responsible for using tessellation factors (computed in the HS stage) to tessellate U,V parametric domains into domain point topologies.
Domain Shader (DS)	The Domain Shader stage is responsible for processing (shading) the domain points (generated by the TE stage) into corresponding vertices.
Geometry Shader (GS)	The Geometry Shader stage is responsible for processing incoming objects by passing each object's vertices to a GS thread.
Stream Output Logic (SOL)	The Stream Output Logic is responsible for outputting incoming object vertices into Stream Out Buffers in memory.
Clipper (CLIP)	The Clipper stage performs Clip Tests on incoming objects and clips objects if required. Objects are clipped using fixed-function hardware.
Strip/Fan (SF)	The Strip/Fan stage performs object setup. Object setup uses fixed-function hardware.

Pipeline Stage	Functions Performed
Windower/Masker (WM)	The Windower/Masker performs object rasterization and determines visibility coverage.

3D Pipeline-Level State

This section contains table commands for the 3D Pipeline Level.

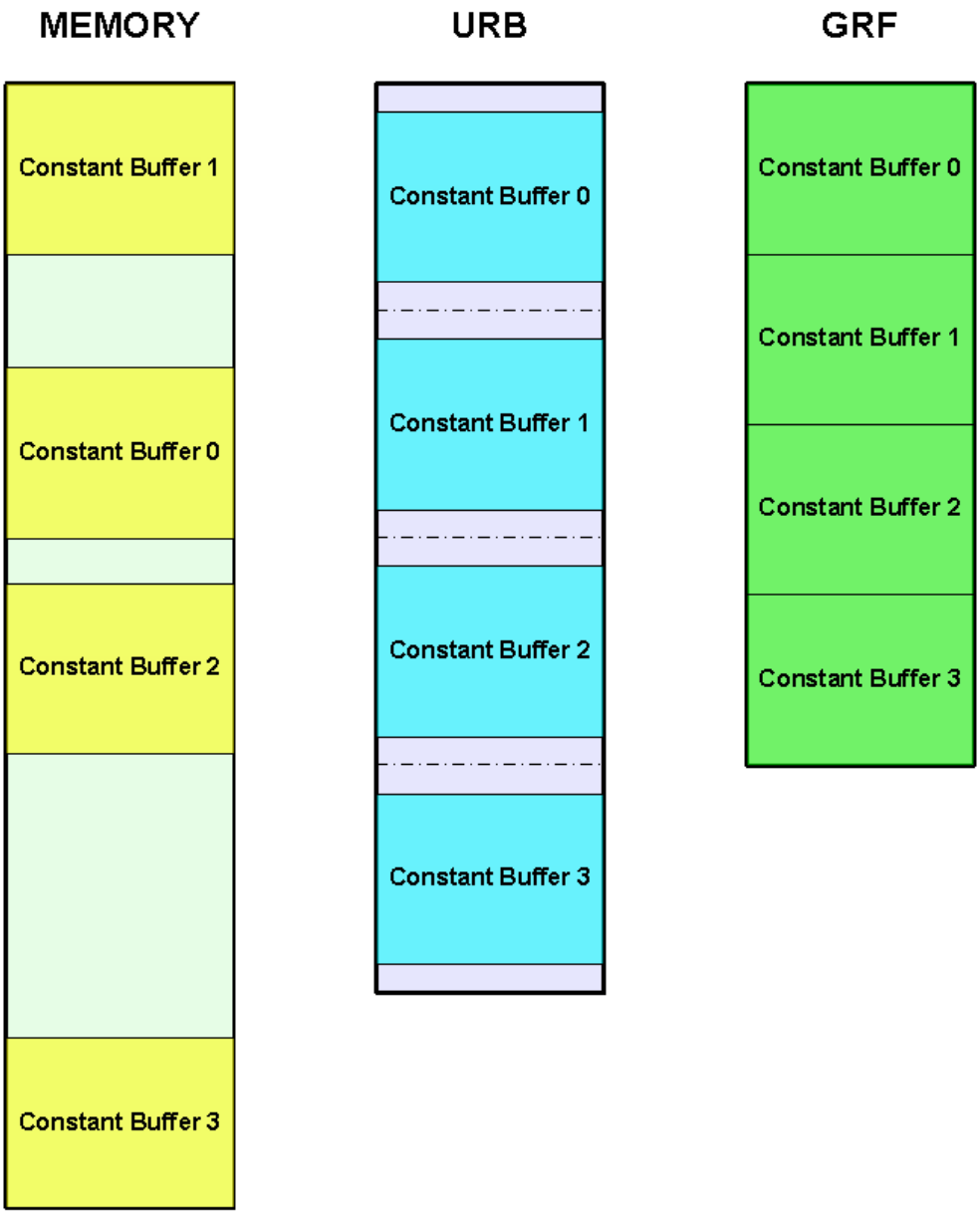
Programming Note	
Context:	3D Pipeline-Level State - Push Constant URB Allocation

The push constants are buffered in the Push Constant section of the URB which is part of the L3$. Software is required to program the hardware to allocate space in the URB for each shader push constant. The software is limited to the low addresses of the URB and must ensure that none of the shaders have overlapping handles.

Software may use some if not all of the Push Constant region of the URB for pr-stage handle allocations as long as none of the push constants and handle allocations overlap.

Refer to the various 3DSTATE_PUSH_CONSTANT_ALLOC_xx state commands for details regarding the maximum size of the Push Constant and other state programming information.

Below is a diagram that represents how the hardware may move and store one CONSTANT_BUFFER command for a fixed function shader:



The bubbles in the URB are caused by the constant buffer in memory starting on a half cacheline and being an even number in length. If the constant buffer starts on an odd cacheline and has an odd number length then there will only be a bubble at the beginning of the buffer in the URB. If the constant buffer in memory starts on a cache line boundary and has an odd number length then the bubble will only be at the end of the constant buffer in the URB. Once the constant buffer is written to the GRF space then all the bubbles will be removed.

Software must guarantee that there is enough space in the push constant buffer in the URB to hold one constant buffer from memory. This includes any buffering to write the 512b aligned requests from memory into the URB. Because the L3$ only supports writes from memory in 512b chunks, the URB may have some bubbles between each constant buffer fetch.

Statistics

This topic is currently under development.

Statistics Gathering

The table below describes how the device supports the required API statistics counters.

API-level Statistic	HW Support
IAVertices = # of vertices IA generated. May or may not include (a) vertices in partial primitives, (b) unused adjacent-only vertices. Not affected by vertex caching.	VF maintains **IA_VERTICES_COUNT**. Will include unused adjacent-only vertices. Will not include vertices in partial primitives.
IAPrimitives = # of primitives (objects) IA generated. May or may not include partial primitives.	VF maintains **IA_PRIMITIVES_COUNT**. Will not include partial primitives. Will not count patch topologies that do not match what the HS or GS expects as input , if enabled (i.e., mismatching patch topologies are discarded by VF).
VSInvocations = # of times VS is executed. May be affected by vertex caching. May or may not include (a) shared vertices in non-indexed strips, (b) vertices in partial primitives, (c) unused adjacent-only vertices.	VS maintains **VS_INVOCATION_COUNT**. Impacted by vertex caching. Will not include vertices in partial primitives. Will include unused adjacent-only vertices. Will not include shared vertices in non-indexed strips, unless pre-empted. Increments even if VS Function Enable is DISABLED.
HSInvocations = # of patches executed by HS.	HS maintains **HS_INVOCATION_COUNT**. This gets incremented by 1 for each patch whenever HS is enabled.
DSInvocations = # of times DS is executed to shade a domain point. Allows HW to shade identical domain points multiple times, with the exception of point outputs where only unique domain points can be generated.	DS maintains **DS_INVOCATION_COUNT**. This is incremented for each domain point passed to a DS thread.
GSInvocations = # of times GS is executed. Obviously does not include partial primitives. May be incremented when StreamOut enabled, even if NULL_GS.	GS maintains **GS_INVOCATION_COUNT**, incrementing it by **GSInvocations Increment Value** for each dispatched instance. Will not be incremented if NULL_GS.
GSPrimitives = # of primitives GS generated. Does not include primitives passing through a disabled GS stage. May or may not include partial primitives output by GS.	GS maintains **GS_PRIMITIVE_COUNT**. GS unit will increment this as it parses the GS thread output. Will not include partial primitives output by GS

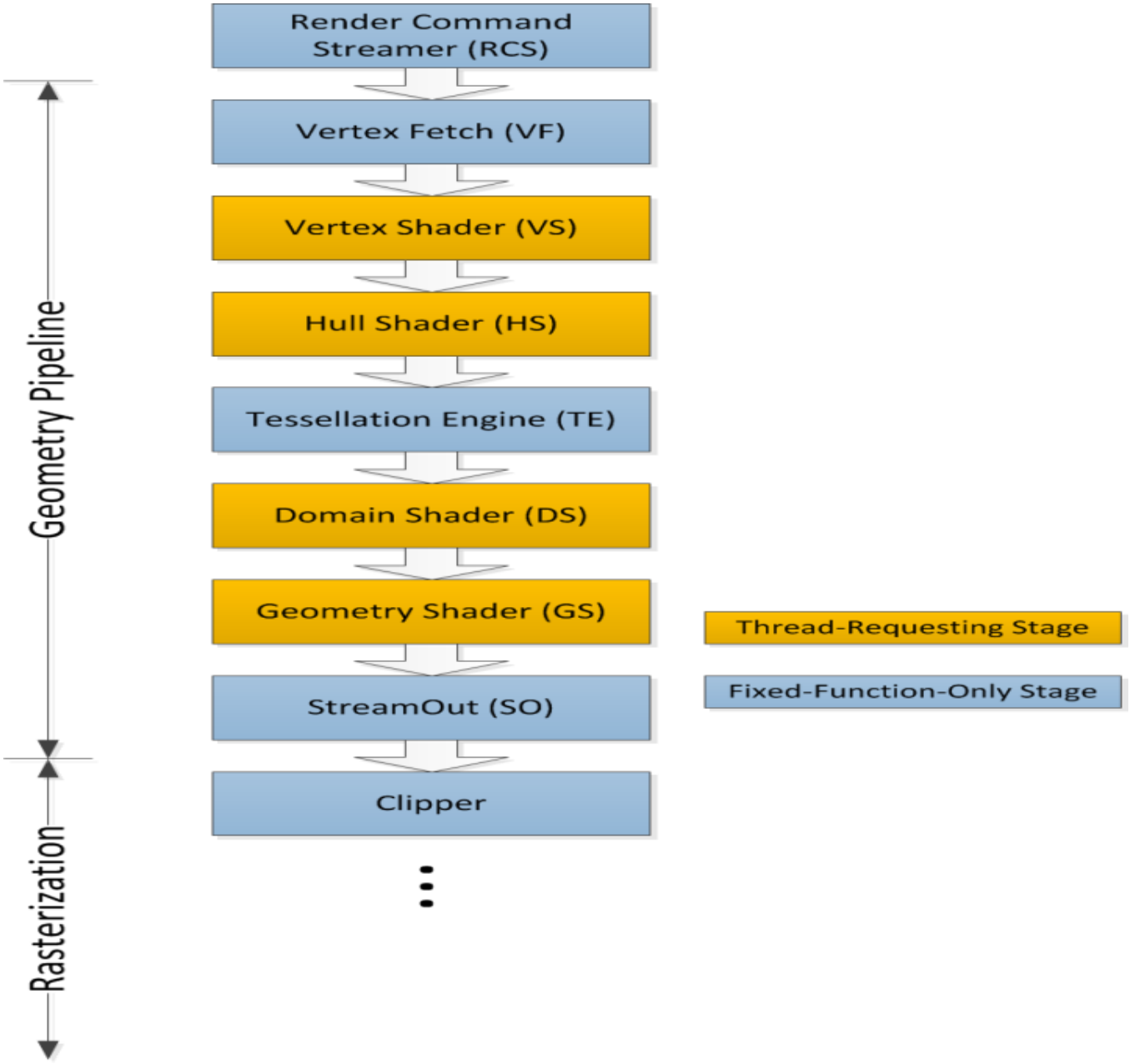
API-level Statistic	HW Support
	threads.
NumPrimitivesWritten[<stream#>] = # of complete primitives written to the stream's SO buffer, subject to buffer overflow.	SOL maintains **SO_NUM_PRIMS_WRITTEN[0-3]**.
PrimitiveStorageNeeded[<stream#>] = # of complete primitives which would have been written to the stream's SO buffer ignoring any overflow.	SOL maintains **SO_PRIM_STORAGE_NEEDED[0-3]**.
CInvocations = # of primitives entering rasterization (which starts with the clipper) and isn't affected by any actual clipping. Does not increment when rasterization is disabled (e.g., when StreamOut is the last enabled stage). May or may not include partial primitives.	CL OSB maintains **CL_INVOCATION_COUNT**. Will not include partial primitives. Note that the SOL (regardless of SO enabled) will discard primitives if rendering is disabled, so these primitives will not reach the CL unit.
CPrimitives = # of primitives output from clipper. I.e., doesn't increment if TrivReject or dropped due to NaNs, increments by 1 if TrivAccept, or increments by number of primitives generated if MustClip. Does not increment when rasterization is disabled. May or may not include partial primitives. Accomodates infinite or no guardband.	SF OSB maintains **CL_PRIMITIVES_COUNT**. Will not include partial primitives.
PSInvocations = # of times PS is executed, including unlit "helper pixels" within a subspan that need to go through the PS shader to provide 2x2 gradients. Accomodates early depth/stencil. Does not increment if NULL PS. Multisampling: counts pixels shaded If PERPIXEL or samples shaded if PERSAMPLE.	WIZ maintains **PS_INVOCATION_COUNT**.
Occlusion = # of "visible" multisamples which passed both depth and stencil testing. Doesn't include PS-discarded pixels or oMask/AlphaToCoverage-killed samples. Both (a) a disabled test (depth or stencil) and (b) no bound RT or Depth/Stencil buffer conditions count as always passing.	WIZ & PBE maintain **PS_DEPTH_COUNT**.

3D Pipeline Geometry

This topic is currently under development.

Block Diagram

The following block diagram shows the stages of the Geometry Pipeline and where they are positioned in the overall 3D Pipeline.



3D Primitives Overview

The 3DPRIMITIVE command (defined in the VF Stage chapter) is used to submit 3D primitives to be processed by the 3D pipeline. Typically the processing results in the rendering of pixel data into the render targets, but this is not required.

There is considerable confusion surrounding the term 'primitive', e.g., is a triangle strip a 'primitive', or is a triangle within a triangle strip a 'primitive'? Some APIs use the term 'topology' to describe the higher-level construct (e.g., a triangle strip), and uses the term 'primitive' when discussing a triangle within a triangle strip. In this spec, we will try to avoid ambiguity by using the term 'object' to represent the basic shapes (point, line, triangle), and 'topology' to represent input geometry (strips, lists, etc.). Unfortunately, terms like '3DPRIMITIVE' must remain for legacy reasons.

The following table describes the basic primitive topology types supported in the 3D pipeline.

Programming Note	
Project:	All
Context:	3D Primitives Overview

- There are several variants of the basic topologies. These have been introduced to allow slight variations in behavior without requiring a state change.
- Number of vertices and Dangling Vertices: Topologies have an "expected" number of vertices in order to form complete objects within the topologies (e.g., LINELIST is expected to have an even number of vertices). The actual number of vertices specified in the 3DPRIMITIVE command, and as output from the GS unit, is allowed to deviate from this expected number, in which case any "dangling" vertices are discarded. The removal of dangling vertices is initially performed in the VF unit. To filter out dangling vertices emitted by GS threads, the CLIP unit also performs dangling-vertex removal at its input.

3D Primitive Topology Types

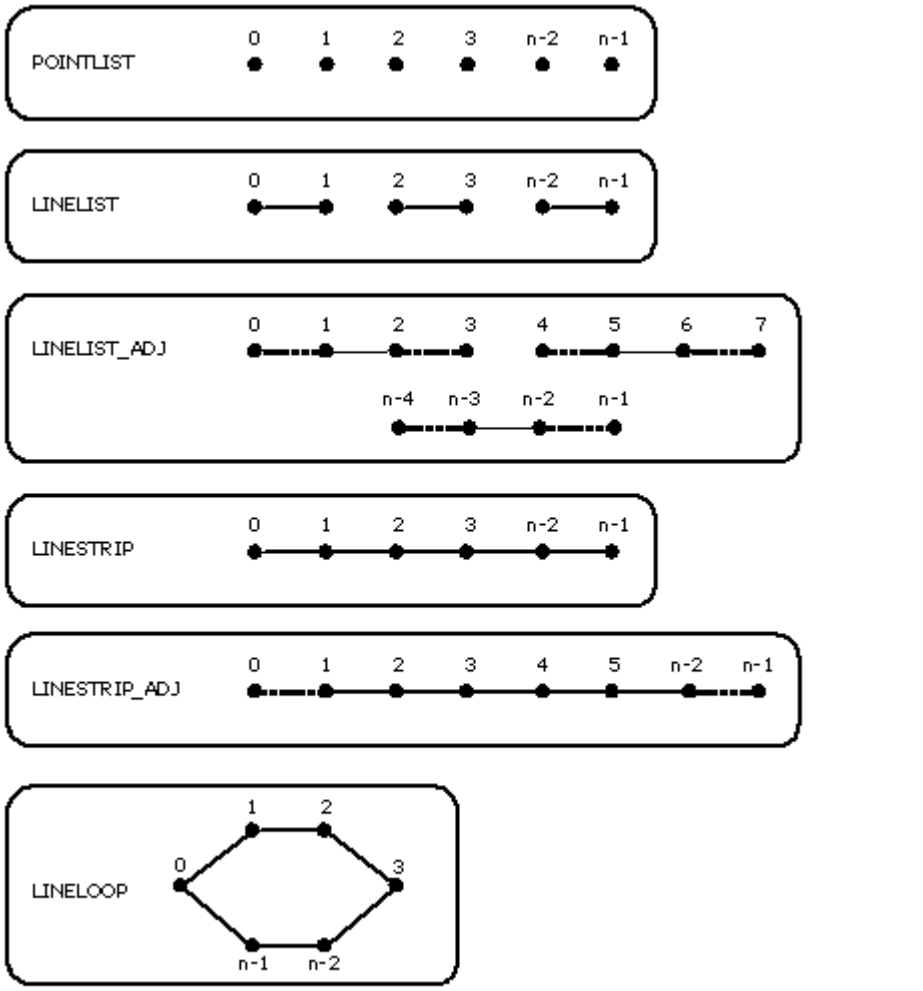
3D Primitive Topology Type (ordered alphabetically)	Description
LINELIST	• A list of independent line objects (2 vertices per line). • Normal usage expects a multiple of 2 vertices, though incomplete objects are silently ignored.
LINELIST_ADJ	• A list of independent line objects with adjacency information (4 vertices per line). • Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
LINELOOP	• Similar to a 3DPRIM_LINESTRIP, though the last vertex is connected back to the initial vertex via a line object. The LINELOOP topology is converted to LINESTRIP topology at the beginning of the 3D pipeline. • Normal usage expects at least 2 vertices, though incomplete objects are silently ignored. (The 2-vertex case is required by OGL). • Not valid after the GS stage (i.e., must be converted by a GS thread to some other primitive type).
LINESTRIP	• A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define a connected line object. • Normal usage expects at least 2 vertices, though incomplete objects are silently ignored.
LINESTRIP_ADJ	• A list of vertices connected such that, after the first vertex, each additional vertex is associated with the previous vertex to define connected line object. The first and last segments are adjacent–only vertices. • Normal usage expects at least 4 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
LINESTRIP_BF	• Similar to LINESTRIP, except treated as “backfacing’ during rasterization (stencil test). • This can be used to support “line” polygon fill mode when two-sided stencil is enabled.
LINESTRIP_CONT	• Similar to LINESTRIP, except LineStipple (if enabled) is continued (vs. reset) at the start of the primitive topology. • This can be used to support line stipple when the API-provided primitive is split across multiple tolopologies.

3D Primitive Topology Type (ordered alphabetically)	Description
LINESTRIP_CONT_BF	Combination of LINESTRIP_BF and LINESTRIP_CONT variations.
POINTLIST	A list of point objects (1 vertex per point).
POINTLIST_BF	• Similar to POINTLIST, except treated as "backfacing' during rasterization (stencil test). • This can be used to support "point" polygon fill mode when two-sided stencil is enabled.
POLYGON	• Similar to TRIFAN, though the first vertex always provides the "flat-shaded" values (vs. this being programmable through state). • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
QUADLIST	• A list of independent quad objects (4 vertices per quad). • The QUADLIST topology is converted to POLYGON topology at the beginning of the 3D pipeline. • Normal usage expects a multiple of 4 vertices, though incomplete objects are silently ignored.
QUADSTRIP	• A list of vertices connected such that, after the first two vertices, each additional pair of vertices are associated with the previous two vertices to define a connected quad object. • Normal usage expects an even number (4 or greater) of vertices, though incomplete objects are silently ignored.
RECTLIST	• A list of independent rectangles, where only 3 vertices are provided per rectangle object, with the fourth vertex implied by the definition of a rectangle. V0=LowerRight, V1=LowerLeft, V2=UpperLeft. Implied V3 = V0-V1+V2. • Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored. • The RECTLIST primitive is supported specifically for 2D operations (e.g., BLTs and "stretch" BLTs) and not as a general 3D primitive. Due to this, a number of restrictions apply to the use of RECTLIST: • Must utilize "screen space" coordinates (VPOS_SCREENSPACE) when the primitive reaches the CLIP stage. The W component of position must be 1.0 for all vertices. The 3 vertices of each object should specify a screen-aligned rectangle (after the implied vertex is computed). • Clipping: Must not require clipping or rely on the CLIP unit's ClipTest logic to determine if clipping is required. Either the CLIP unit should be DISABLED, or the CLIP unit's Clip Mode should be set to a value other than CLIPMODE_NORMAL.

3D Primitive Topology Type (ordered alphabetically)	Description
	• Viewport Mapping must be DISABLED (as is typical with the use of screen-space coordinates).
RECTLIST_SUBPIXEL	The subpixel precise, axis-aligned bounding box of the object's 3 vertices is rendered.
TRIFAN	• Triangle objects arranged in a fan (or polygon). The initial vertex is maintained as a common vertex. After the second vertex, each additional vertex is associated with the previous vertex and the common vertex to define a connected triangle object. • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
TRIFAN_NOSTIPPLE	• Similar to TRIFAN, but poylgon stipple is not applied (even if enabled). • This can be used to support "point" polygon fill mode, under the combination of the following conditions: (a) when the frontfacing and backfacing polygon fill modes are different (so the final fill mode is not known to the driver) (b) one of the fill modes is "point" and the other is "solid" (c) point mode is being emulated by converting the point into a trifan (d) polygon stipple is enabled. In this case, polygon stipple should not be applied to the points-emulated-as-trifans.
TRILIST	• A list of independent triangle objects (3 vertices per triangle). • Normal usage expects a multiple of 3 vertices, though incomplete objects are silently ignored.
TRILIST_ADJ	• A list of independent triangle objects with adjacency information (6 vertices per triangle). • Normal usage expects a multiple of 6 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
TRISTRIP	• A list of vertices connected such that, after the first two vertices, each additional vertex is associated with the last two vertices to define a connected triangle object. • Normal usage expects at least 3 vertices, though incomplete objects are silently ignored.
TRISTRIP_ADJ	• A list of vertices where the even-numbered (including 0th) vertices are connected such that, after the first two vertex pairs, each additional even-numbered vertex is associated with the last two even-numbered vertices to

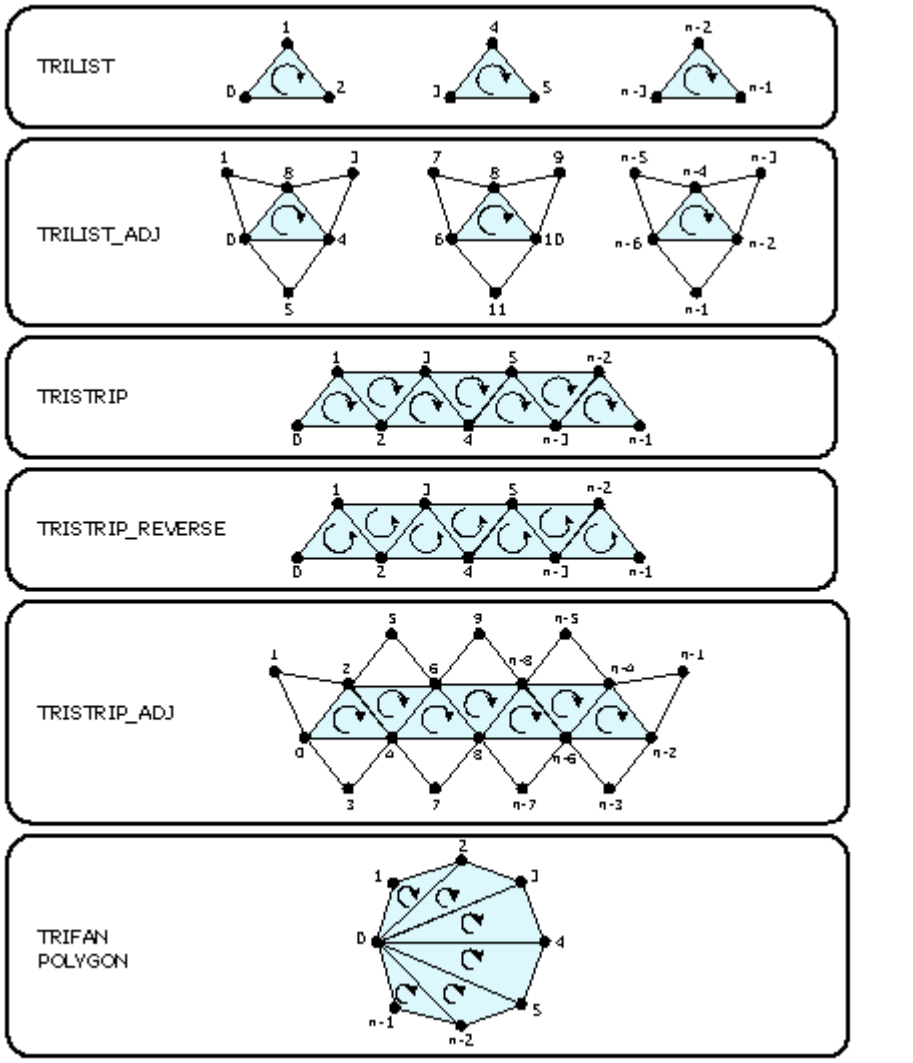
3D Primitive Topology Type (ordered alphabetically)	Description
	define a connected triangle object. The odd-numbered vertices are adjacent-only vertices. VFUNIT will complete a drawcall with the topology of tristrip_adj even if there is a preemption request in the middle of the draw call. • Normal usage expects at least 6 vertices, though incomplete objects are silently ignored. • Not valid as output from GS thread.
TRISTRIP_REVERSE	Similar to TRISTRIP, though the sense of orientation (winding order) is reversed – this allows SW to break long tristrips into smaller pieces and still maintain correct face orientations.
[CHV, BSW]: PATCHLIST_n	List of n-vertex "patch" objects. These topologies cannot be rendered directly – the tessellation units must be used to convert them into points, lines, or triangles to produce rasterization results. (VS, GS, and StreamOutput operations can also be performed.)

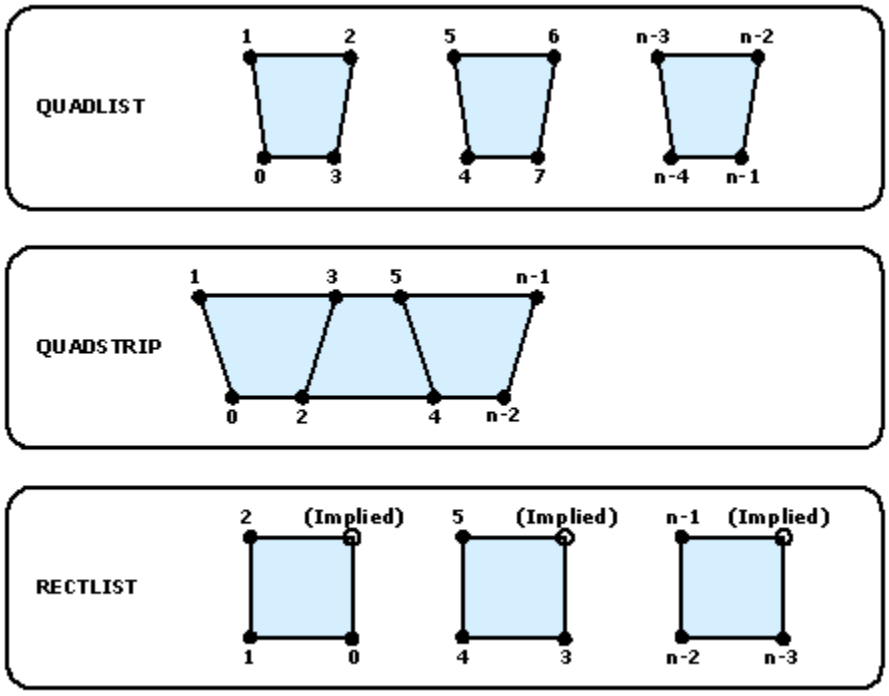
The following diagrams illustrate the basic 3D primitive topologies. (Variants are not shown if they have the same definition with respect to the information provided in the diagrams).



B6815-01

A note on the arrows you see below: These arrows are intended to show the vertex ordering of triangles that are to be considered having “clockwise” winding order in screen space. Effectively, the arrows show the order in which vertices are used in the cross-product (area, determinant) computation. Note that for TRISTRIP, this requires that either the order of odd-numbered triangles be reversed in the cross-product or the sign of the result of the normally-ordered cross-product be flipped (these are identical operations).


TRILIST
TRILIST_ADJ
TRISTRIP
TRISTRIP_REVERSE
TRISTRIP_ADJ
TRIFAN
POLYGON
B6816-01


QUADLIST
1
2
0
3
5
6
4
7
n-3
n-2
n-4
n-1
QUADSTRIP
1
3
5
n-1
0
2
4
n-2
RECTLIST
2
(Implied)
1
0
5
(Implied)
4
3
n-1
(Implied)
n-2
n-3

B6818-01

Vertex Data Overview

The 3D pipeline FF stages (past VF) receive input 3D primitives as a stream of vertex information packets. (These packets are not directly visible to software.) Much of the data associated with a vertex is passed indirectly via a VUE handle. The information provided in vertex packets includes:

- The **URB Handle** of the VUE: This is used by the FF unit to refer to the VUE and perform any required operations on it (e.g., cause it to be read into the thread payload, dereference it, etc.).
- **Primitive Topology Information:** This information is used to identify/delineate primitive topologies in the 3D pipeline. Initially, the VF unit supplies this information, which then passes through the VS stage unchanged. GS and CLIP threads must supply this information with each vertex they produce (via the URB_WRITE message). If a FF unit directly outputs vertices (that were not generated by a thread they spawned), that FF unit is responsible for providing this information.
  - **PrimType:** The type of topology, as defined by the corresponding field of the 3DPRIMITIVE command.
  - **StartPrim:** TRUE only for the first vertex of a topology.
  - **EndPrim:** TRUE only for the last vertex of a topology.
- (Possibly, depending on FF unit) Data read back from the **Vertex Header** of the VUE.

Vertex Positions

(For brevity, the following discussion uses the term map as a shorthand for “compute screen space coordinate via perspective divide followed by viewport transform”.)

The “Position” fields of the Vertex Header are the only vertex position coordinates exposed to the 3D Pipeline. The CLIP and SF units are the only FF units which perform operations using these positions. The VUE will likely contain other position attributes for the vertex outside of the Vertex Header, though this information is not directly exposed to the FF units. For example, the Clip Space position will likely be required in the VUE (outside of the Vertex Header) to perform correct and robust 3D Clipping in the CLIP thread.

CLIP unit uses the **3DSTATE_CLIP.PerspectiveDivideDisable** bit to deteremine whether to perform a perspective projection (divide by w) of the read-back 4D Position.

When Perspective Divide is enabled, the Clip Space position is defined in a homogeneous 4D coordinate space (pre-perspective divide), where the visible “view volume” is defined by the APIs. The API’s VS, GS or DS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a “view transform” in this computation path). When Perspective Divide is enabled, the 3D FF pipeline will perform a perspective projection (division of x,y,z by w), perform clip-test on the resulting NDC (Normalized Device Coordinates), and eventually perform viewport mapping (in the SF unit) to yield screen-space (pixel) coordinates.

When Perspective Divide is disabled, the read-back Position does not undergo perspective projection by the 3D FF pipeline.

Clip Space Position

The *clip-space* position of a vertex is defined in a homogeneous 4D coordinate space where, after perspective projection (division by W), the visible "view volume" is some canonical (3D) cuboid. Typically the X/Y extents of this cuboid are [-1,+1], while the Z extents are either [-1,+1] or [0,+1]. The API's VS or GS shader program will include geometric transforms in the computation of this clip space position such that the resulting coordinate is positioned properly in relation to the view volume (i.e., it will include a "view transform" in this computation path).

Note that, under typical perspective projections, the clip-space W coordinate is equal to the view-space Z coordinate.

A vertex's clip-space coordinates must be maintained in the VUE up to 3D clipping, as this clipping is performed in clip space.

In [CHV, BSW], vertex clip-space positions must be included in the Vertex Header, so that they can be read-back (prior to Clipping) and then subjected to perspective projection (in hardware) and subsequent use by the FF pipeline.

NDC Space Position

A perspective divide operation performed on a clip-space position yields a [X,Y,Z,RHW] NDC (Normalized Device Coordinates) space position. Here "normalized" means that visible geometry is located within the [-1,+1] or [0,+1] extent view volume cuboid (see clip-space above).

- The NDC X,Y,Z coordinates are the clip-space X,Y,Z coordinates (respectively) divided by the clip-space W coordinate (or, more correctly, the clip-space X,Y,Z coordinates are multiplied by the reciprocal of the clip space W coordinate).
    - Note that the X,Y,Z coordinates may contain INFINITY or NaN values (see below).
- The NDC RHW coordinate is the reciprocal of the clip-space W coordinate and therefore, under normal perspective projections, it is the reciprocal of the view-space Z coordinate. Note that NDC space is really a 3D coordinate space, where this RHW coordinate is retained in order to perform perspective-correct interpolation, etal. Note that, under typical perspective projections.
    - Note that the RHW coordinate make contain an INFINITY or NaN value (see below).

Screen-Space Position

Screen-space coordinates are defined as:

- X,Y coordinates are in absolute screen space (pixel coordinates, upper left origin). See Vertex X,Y Clamping and Quantization in the SF section for a discussion of the limitations/restrictions placed on screenspace X,Y coordinates.
- Z coordinate has been mapped into the range used for DepthTest.
- RHW coordinate is actually the reciprocal of clip-space W coordinate (typically the reciprocal of the view-space Z coordinate).

Vertex Fetch (VF) Stage

The Vertex Fetch Stage performs one major function: executing 3DPRIMITIVE commands. This is handled by the VF's InputAssembly function.

State

This section contains various state registers.

Control State

3DSTATE_VF

3DSTATE_VF_TOPOLOGY

Index Buffer (IB) State

The 3DSTATE_INDEX_BUFFER command is used to define an *Index Buffer* (IB) used in subsequent 3DPRIMITIVE commands.

The RANDOM access mode of the 3DPRIMITIVE command involves the use of a memory-resident IB. The IB, defined via the 3DSTATE_INDEX_BUFFER command described below, contains a 1D array of 8, 16 or 32-bit index values. These index values will be fetched by the InputAssembly function, and subsequently used to compute locations in VERTEXDATA buffers from which the actual vertex data is to be fetched. (This is opposed to the SEQUENTIAL access mode were the vertex data is simply fetched sequentially from the buffers).

The following table lists which primitive topology types support the presence of Cut Indices.

Definition	Cut Index?
3DPRIM_POINTLIST	Y
3DPRIM_LINELIST	Y
3DPRIM_LINESTRIP	Y
3DPRIM_TRILIST	Y
3DPRIM_TRISTRIP	Y
3DPRIM_LINELIST_ADJ	Y
3DPRIM_LINESTRIP_ADJ	Y
3DPRIM_TRILIST_ADJ	Y
3DPRIM_TRISTRIP_ADJ	Y
3DPRIM_TRISTRIP_REVERSE	Y
3DPRIM_RECTLIST	N
3DPRIM_POINTLIST_BF	Y
3DPRIM_LINESTRIP_CONT	Y
3DPRIM_LINESTRIP_BF	Y
3DPRIM_LINESTRIP_CONT_BF	Y

Definition	Cut Index?
3DPRIM_TRIFAN_NOSTIPPLE	N

3DSTATE_INDEX_BUFFER

Vertex Buffers (VB) State

The 3DSTATE_VERTEX_BUFFERS and 3DSTATE_INSTANCE_STEP_RATE commands are used to define *Vertex Buffers* (VBs) used in subsequent 3DPRIMITIVE commands.

Most input vertex data is sourced from memory-resident VBs. A VB is a 1D array of structures, where the size of the structure as defined by the VB's **BufferPitch**. VBs are accessed either as *VERTEXDATA buffers* or *INSTANCEDATA buffers*, as defined by the VB's **BufferAccessType**. The VB's access type will determine whether the VF-computed VertexIndex or InstanceIndex is used to access data in the VB.

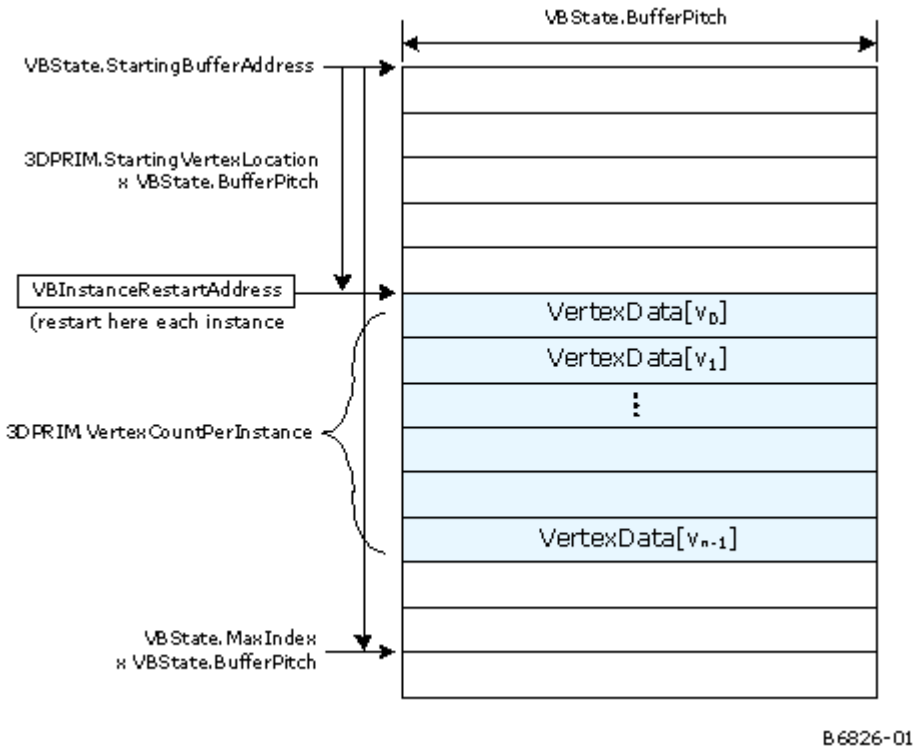
Given that the RANDOM access mode of the 3DPRIMITIVE command utilizes an IB (possibly provided by an application) to compute VB index values, VB definitions contain a **MaxIndex** value used to detect accesses beyond the end of the VBs. Any access outside the extent of a VB returns 0.

3DSTATE_VERTEX_BUFFERS

VERTEX_BUFFER_STATE

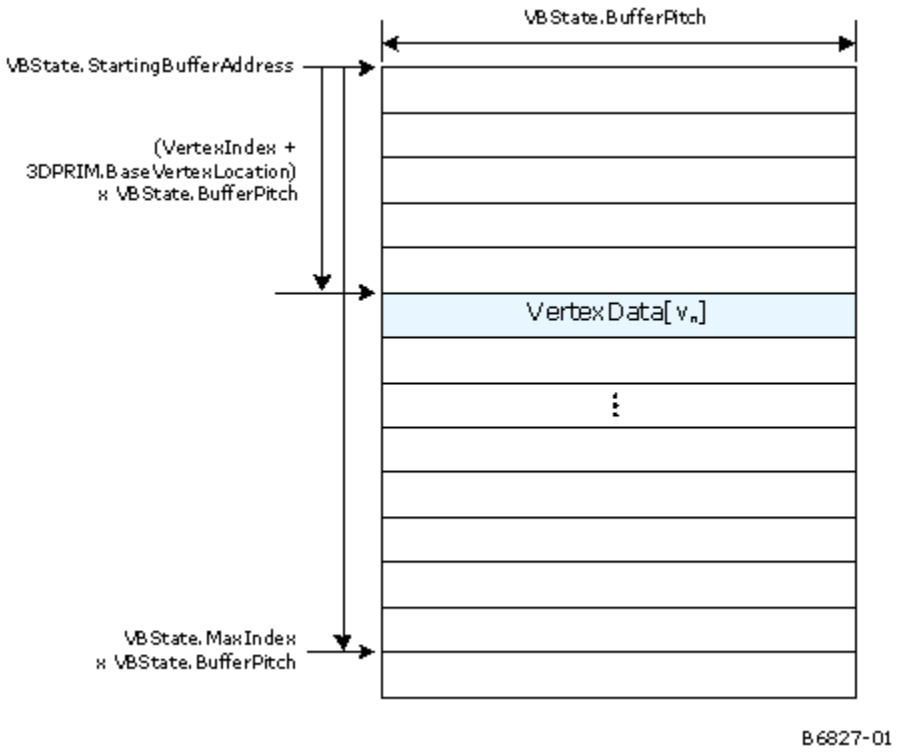
VERTEXDATA Buffers – SEQUENTIAL Access

Description
This section pertains to (a) 3DPRIMITIVE commands with VertexAccessType = SEQUENTIAL and (b) vertex elements with InstancingEnable set to DISABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



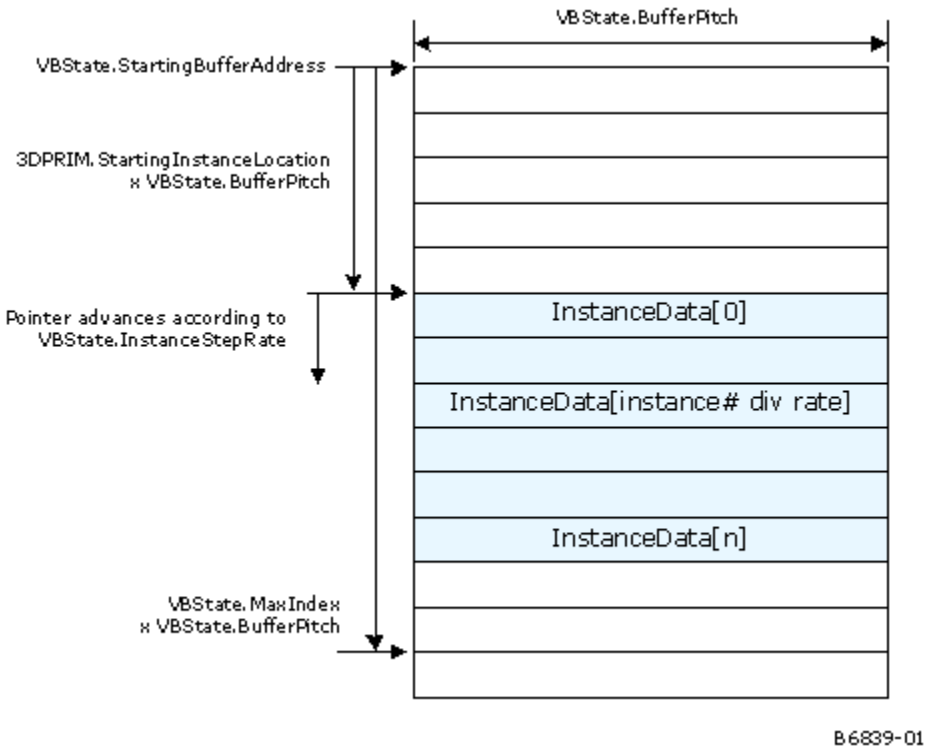
VERTEXDATA Buffers – RANDOM Access

Description
This section pertains to (a) 3DPRIMITIVE commands with VertexAccessType = RANDOM and (b) vertex elements with InstancingEnable set to DISABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



INSTANCEDATA Buffers

Description
This section pertains to vertex elements with InstancingEnable set to ENABLED. Instead of "VBState.StartingBufferAddress + VBState.MaxIndex x VBState.BufferPitch", the address of the byte immediately beyond the last valid byte of the buffer is determined by "VBState.StartingBufferAddress + VBState.BufferSize".



Vertex Definition State

The following subsections define the state information for vertex data and describe some related processing.

Input Vertex Definition

The 3DSTATE_VERTEX_ELEMENTS command is used to define the source and format of input vertex data and the format of how it is stored in the destination VUE as part of 3DPRIMITIVE processing in the VF unit.

Two additional commands are added. 3DSTATE_VF_INSTANCING specifies the InstanceStepRate on a per-vertex-element basis. 3DSTATE_VF_SGVS specifies optional insertion of VertexID and/or InstanceID into the input vertex data (logically following the processing of the VERTEX_ELEMENT_STATE structures).

Refer to *3DPRIMITIVE Processing* below for the general flow of how input vertices are input and stored during processing of the 3DPRIMITIVE command.

VERTEX_ELEMENT_STATE

3DSTATE_VERTEX_ELEMENTS

3D_Vertex_Component_Control

3DSTATE_VF_INSTANCING

3DSTATE_VF_SGVS

3D Primitive Command

Following are 3D Primitive Commands:

3DPRIMITIVE

3D Primitive Topology Type Encoding

The following table defines the encoding of the Primitive Topology Type field. See *3D Pipeline* for details, programming restrictions, diagrams, and a discussion of the basic primitive types.

3D_Prim_Topo_Type

Functions

This section covers the various functions for Vertex Fetch.

Input Assembly

The VF's InputAssembly function includes (for each vertex generated):

- Generation of VertexIndex and InstanceIndex for each vertex, possibly via use of an Index Buffer.
- Lookup of the VertexIndex in the Vertex Cache (if enabled)
- If a cache miss is detected:
  - Use of computed indices to fetch data from memory-resident vertex buffers
  - Format conversion of the fetched vertex data
  - Assembly of the format conversion results (and possibly some internally generated data) to form the complete "input" (raw) vertex
  - Storing the input vertex data in a Vertex URB Entry (VUE) in the URB
  - Output of the VUE handle of the input vertex to the VS stage
- If a cache hit is detected, the VUE handle from the Vertex Cache is passed to the VS stage (marked as a cache hit to prevent any VS processing).

Vertex Assembly

The VF utilizes a number of VERTEX_ELEMENT state structures to define the contents and format of the vertex data to be stored in Vertex URB Entries (VUEs) in the URB. See below for a detailed description of the command used to define these structures (3DSTATE_VERTEX_ELEMENTS).

Each active VERTEX_ELEMENT structure defines up to 4 contiguous DWords of VUE data, where each DWord is considered a "component" of the vertex element. The starting destination DWord offset of the vertex element in the VUE is specified, and the VERTEX_ELEMENT structures must be defined with monotonically increasing VUE offsets. For each component, the source of the component is specified. The source may be a constant (0, 0x1, or 1.0f), a generated ID (VertexID, InstanceID or PrimitiveID), or a component of a structure in memory (e.g,. the Y component of an XYZW position in memory). In the case of a memory source, the Vertex Buffer sourcing the data, and the location and format of the source data with that VB are specified.

The VF's Vertex Assembly process can be envisioned as the VF unit stepping through the VERTEX_ELEMENT structures in order, fetching and format-converting the source information (if memory resident), and storing the results in the destination VUE.

The information supplied via the 3DSTATE_VF_SGVS command is also used to optionally insert VertexID and/or InstanceID into the input vertex data, after the VERTEX_ELEMENT structures are processed.

Vertex Cache

The VF stage communicates with the VS stage in order to implement a Vertex Cache function in the 3D pipeline. The Vertex Cache is strictly a performance-enhancing feature and has no impact on 3D pipeline results (other than a few statistics counters).

The Vertex Cache contains the VUE handles of VS-output (shaded) vertices if the VS function is enabled, and the VUE handles of VF-output (raw) vertices if the VS function is disabled. (Note that the actual vertex data is held in the URB, and only the handles of the vertices are stored in the cache). In either case, the contents of the cache (VUE handles) are tagged with the VertexIndex value used to fetch the input vertex data. The rationale for using the VertexIndex as the tag is that (assuming no other state or parameters change) a vertex with the same VertexIndex as a previous vertex will have the same input data, and therefore the same result from the VF+VS function.

Note that any change to the state controlling the InputAssembly function (e.g., vertex buffer definition), or any change to the state controlling the VS function (if enabled) (e.g., VS kernel), will result in the Vertex Cache being invalidated. In addition, any non-trivial use of instancing (i.e., more than one instance per 3DPRIMITIVE command and the inclusion of instance data in the input vertex) will effectively invalidate the cache between instances, as the InstanceIndex is not included in the cache tag. See Vertex Caching in *Vertex Shader* for more information on the Vertex Cache (e.g., when it is implicitly disabled, etc.)

Modern 3D APIs include the notion of native support for *instanced geometry*. The premise is that the application models a high-level object, and then proceeds to process (render) multiple instances of the object, with each instance including some modification to the data associated with the object. For example, a single chair object could be modeled and rendered multiple times, with each instance using a different position transformation (to place several chairs around a table). That transformation matrix would be considered *instance data*, in that it would be fixed for each instance of the object, changing only between instances. This is opposed to the *vertex data*, which could be unique for each vertex of the object. Note that the instance data is replicated at each vertex as part of the Input Assembly process, and so could also be considered "vertex data" in that respect. In fact, the notion of instance data is confined to the VF stage, and the remainder of the 3D pipeline is unaware of the distinction.

Although the ability to perform instanced geometry ("manually") was available in legacy APIs, modern APIs add support for a single Draw() call to process multiple instances. The application specifies the number of instances, and the number of vertices in the instanced object (instance) in addition to other parameters discussed later. The Draw() support effectively becomes a nested loop, with the outer loop dealing with instance variables, and the inner loop sequencing through the object vertices.

Given the flexibility and programmability of the 3D pipeline, the division between vertex and instance data is arbitrary. The application simply marks input data buffers as either containing vertex or instance data. In the outer instance loop, the data specified as coming from *instance buffers* is (effectively) fetched/generated and then associated with each vertex of the instance in the inner loop and combined with the vertex data. New instance data is then gathered for the next instance (the "stepping" of instance data is specified on a per-buffer basis, and different portions of the instance data are allowed to step at different rates -- refer to details below).

The hardware interface to supply instancing state information is slightly different. Individual vertex elements (instead of buffers) are tagged as instanced or not.

Input Data: Push Model vs. Pull Model

Given the programmability of the pipeline, and the ability of shaders to input (load/sample) data from memory buffers in an arbitrary fashion, the decision arises in whether to push instance/vertex data into the front of the pipeline or defer the data access (pull) to the shaders that require it. Modern APIs directly support the latter model via *auto-generated IDs* in the Input Assembly function. An incrementing *VertexID*, *InstanceID*, and *PrimitiveID* are generated in the Input Assembly process, and these values can be declared as input to the "first enabled, relevant" shader. That shader can, for example, use the HW-generated ID as an index into a memory resource such as a constant buffer or vertex buffer. The 3D pipeline HW supports these IDs as required by the APIs.

There are tradeoffs involved in deciding between these models. For vertex data, it is probably always better to push the data into the pipeline, as the VF hardware attempts to cover the latency of the data fetch. The decision is less clear for instance data, as pushing instance data leads to larger Vertex URB entries which will be holding redundant data (as the instance data for vertices of an object are by definition the same). Regardless, the GEN 3D pipeline supports both models.

Generated IDs

Note that the generated IDs are considered separate from any offset computations performed by the VF unit, and are therefore described separately here.

The VF generates InstanceID, VertexID, and PrimitiveID values as part of the InputAssembly process.

VertexID and InstanceID are only allowed to be inserted into the input vertex data as it is gathered and written into the URB as a VUE.

The definition/use of PrimitiveID is more complicated than the other auto-generated IDs. PrimitiveID is associated with an “object” and not a particular vertex.

Description
It is only available to the GS and HS as a special non-vertex input and the PS as a constant-interpolated attribute. It is not seen by the VS or DS at all.

The PrimitiveID therefore is kept separate from the vertex data. Take for example a TRILIST primitive topology: It should be possible to share vertices between triangles in the list (i.e., reuse the VS output of a vertex), even though each triangle has a different PrimitiveID associated with it.

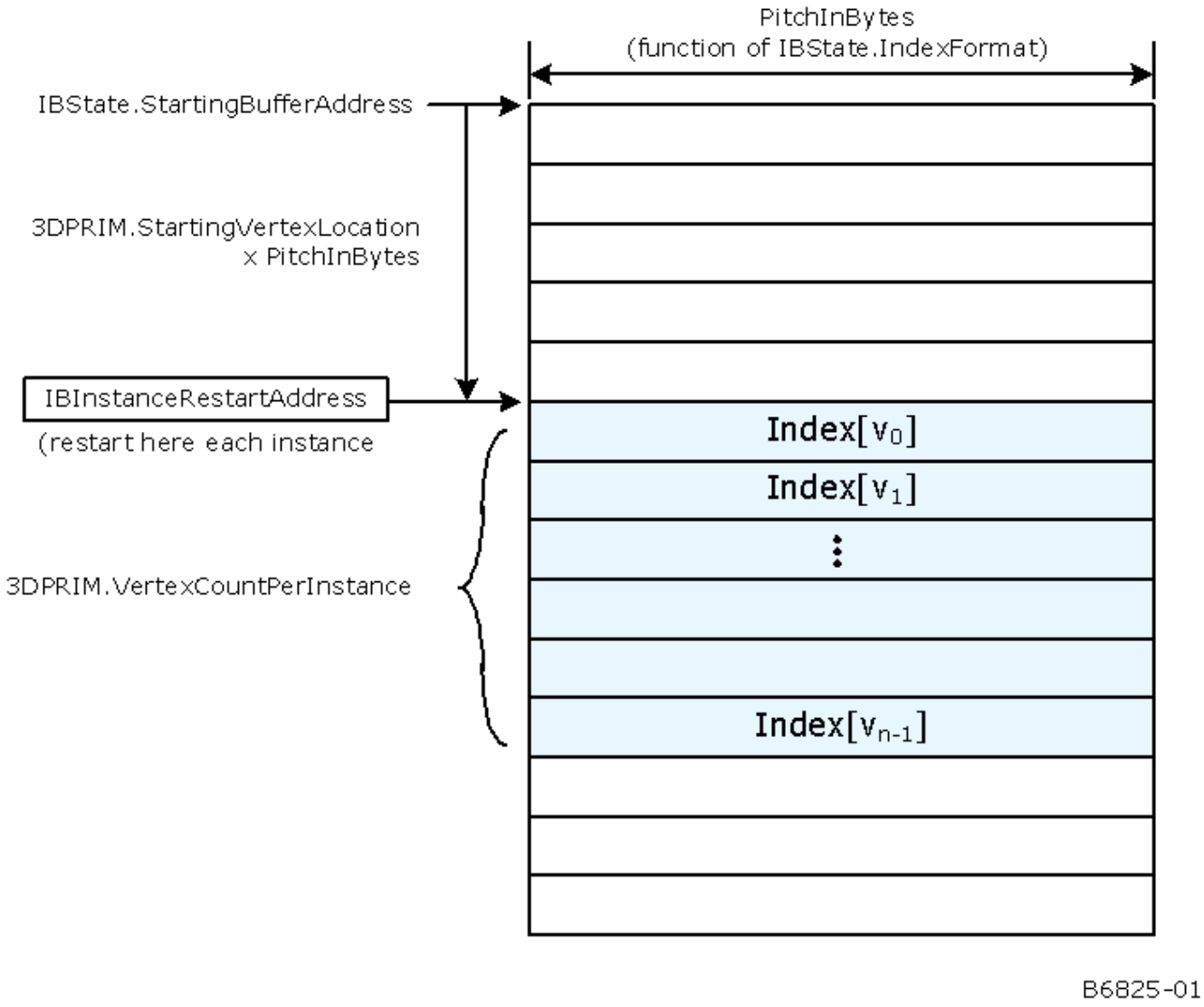
The optional insertion of VertexID and/or InstanceID into the input vertex data occurs as a separate step after the processing of VERTEX_ELEMENT structures, and is controlled via the 3DSTATE_VF_SGVS command.

PrimitiveID is generated by hardware, plumbed down into the HS, GS and SF stages. It is passed along in HS/GS thread payloads. Software can also select PrimitiveID to be swizzled into vertex attribute data in the SF stage, though only if neither the HS nor GS stages are enabled.

3D Primitive Processing

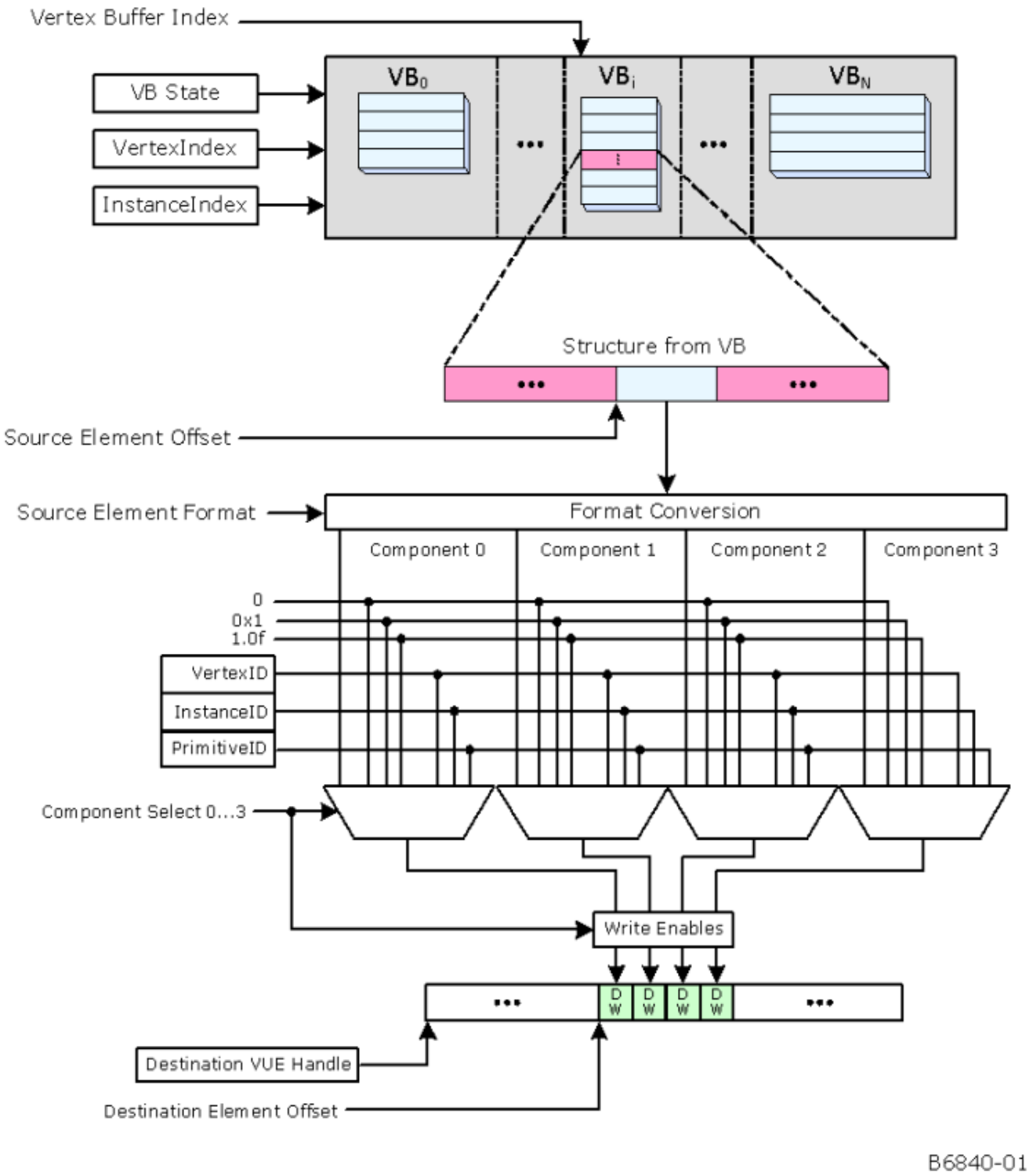
Index Buffer Access

The following figure illustrates how the Index Buffer is accessed.



Vertex Element Data Path

The following diagram shows the path by which a vertex element within the destination VUE is generated and how the fields of the VERTEX_ELEMENT_STATE structure is used to control the generation.


Vertex Buffer Index
VB State
VertexIndex
InstanceIndex
VB0
VBi
VBN
Structure from VB
Source Element Offset
Source Element Format
Format Conversion
Component 0
Component 1
Component 2
Component 3
0
0x1
1.0f
VertexID
InstanceID
PrimitiveID
Component Select 0...3
Write Enables
D W
Destination VUE Handle
Destination Element Offset
B6840-01

FormatConversion

Once the VE source data has been fetched, it is subjected to format conversion. The output of format conversion is up to 4 32-bit components, each either integer or floating-point (as specified by the **Source Element Format**). See *Sampler* for conversion algorithms.

The following table lists the valid **Source Element Format** selections, along with the format and availability of the converted components (if a component is listed as -, it cannot be used as the source of a VUE component). **Note:** This table is a subset of the list of supported surface formats defined in the *Sampler* chapter. Please refer to that table as the "master list". This table is here only to identify the components available (per format) and their format.

Source Element Formats Supported in VF Unit

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32G32B32A32_FLOAT	FLOAT	R	G	B	A
R32G32B32A32_SINT	SINT	R	G	B	A
R32G32B32A32_UINT	UINT	R	G	B	A
R32G32B32A32_UNORM	FLOAT	R	G	B	A
R32G32B32A32_SNORM	FLOAT	R	G	B	A
R64G64_FLOAT	FLOAT	R	G	-	-
R32G32B32A32_SSCALED	FLOAT	R	G	B	A
R32G32B32A32_USCALED	FLOAT	R	G	B	A
R32G32B32A32_SFIXED	FLOAT	R	G	B	A
R64G64_PASSTHRU	NONE	R	G	-	-
R32G32B32_FLOAT	FLOAT	R	G	B	-
R32G32B32_SINT	SINT	R	G	B	-
R32G32B32_UINT	UINT	R	G	B	-
R32G32B32_UNORM	FLOAT	R	G	B	-
R32G32B32_SNORM	FLOAT	R	G	B	-
R32G32B32_SSCALED	FLOAT	R	G	B	-
R32G32B32_USCALED	FLOAT	R	G	B	-
R32G32B32_SFIXED	FLOAT	R	G	B	-
R16G16B16A16_UNORM	FLOAT	R	G	B	A
R16G16B16A16_SNORM	FLOAT	R	G	B	A
R16G16B16A16_SINT	SINT	R	G	B	A
R16G16B16A16_UINT	UINT	R	G	B	A
R16G16B16A16_FLOAT	FLOAT	R	G	B	A
R32G32_FLOAT	FLOAT	R	G	-	-
R32G32_SINT	SINT	R	G	-	-

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32G32_UINT	UINT	R	G	-	-
R32G32_UNORM	FLOAT	R	G	-	-
R32G32_SNORM	FLOAT	R	G	-	-
R64_FLOAT	FLOAT	R	-	-	-
R16G16B16A16_SSCALED	FLOAT	R	G	B	A
R16G16B16A16_USCALED	FLOAT	R	G	B	A
R32G32_SSCALED	FLOAT	R	G	-	-
R32G32_USCALED	FLOAT	R	G	-	-
R32G32_SFIXED	FLOAT	R	G	-	-
R64_PASSTHRU	NONE	R	-	-	-
B8G8R8A8_UNORM	FLOAT	B	G	R	A
R10G10B10A2_UNORM	FLOAT	R	G	B	A
R10G10B10A2_UINT	UINT	R	G	B	A
R10G10B10_SNORM_A2_UNORM	FLOAT	R	G	B	A
R8G8B8A8_UNORM	FLOAT	R	G	B	A
R8G8B8A8_SNORM	FLOAT	R	G	B	A
R8G8B8A8_SINT	SINT	R	G	B	A
R8G8B8A8_UINT	UINT	R	G	B	A
R16G16_UNORM	FLOAT	R	G	-	-
R16G16_SNORM	FLOAT	R	G	-	-
R16G16_SINT	SINT	R	G	-	-
R16G16_UINT	UINT	R	G	-	-
R16G16_FLOAT	FLOAT	R	G	-	-
B10G10R10A2_UNORM	FLOAT	R	G	B	A
R11G11B10_FLOAT	FLOAT	R	G	B	-
R32_SINT	SINT	R	-	-	-
R32_UINT	UINT	R	-	-	-
R32_FLOAT	FLOAT	R	-	-	-
R32_UNORM	FLOAT	R	-	-	-
R32_SNORM	FLOAT	R	-	-	-
R10G10B10X2_USCALED	FLOAT	R	G	B	-
R8G8B8A8_SSCALED	FLOAT	R	G	B	A
R8G8B8A8_USCALED	FLOAT	R	G	B	A
R16G16_SSCALED	FLOAT	R	G	-	-
R16G16_USCALED	FLOAT	R	G	-	-

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R32_SSCALED	FLOAT	R	-	-	-
R32_USCALED	FLOAT	R	-	-	-
R8G8_UNORM	FLOAT	R	G	-	-
R8G8_SNORM	FLOAT	R	G	-	-
R8G8_SINT	SINT	R	G	-	-
R8G8_UINT	UINT	R	G	-	-
R16_UNORM	FLOAT	R	-	-	-
R16_SNORM	FLOAT	R	-	-	-
R16_SINT	SINT	R	-	-	-
R16_UINT	UINT	R	-	-	-
R16_FLOAT	FLOAT	R	-	-	-
R8G8_SSCALED	FLOAT	R	G	-	-
R8G8_USCALED	FLOAT	R	G	-	-
R16_SSCALED	FLOAT	R	-	-	-
R16_USCALED	FLOAT	R	-	-	-
R8_UNORM	FLOAT	R	-	-	-
R8_SNORM	FLOAT	R	-	-	-
R8_SINT	SINT	R	-	-	-
R8_UINT	UINT	R	-	-	-
R8_SSCALED	FLOAT	R	-	-	-
R8_USCALED	FLOAT	R	-	-	-
R8G8B8_UNORM	FLOAT	R	G	B	-
R8G8B8_SNORM	FLOAT	R	G	B	-
R8G8B8_SSCALED	FLOAT	R	G	B	-
R8G8B8_USCALED	FLOAT	R	G	B	-
R8G8B8_SINT	SINT	R	G	B	-
R8G8B8_UINT	UINT	R	G	B	-
R64G64B64A64_FLOAT	FLOAT	R	G	B	A
R64G64B64_FLOAT	FLOAT	R	G	B	A
R16G16B16_FLOAT	FLOAT	R	G	B	-
R16G16B16_UNORM	FLOAT	R	G	B	-
R16G16B16_SNORM	FLOAT	R	G	B	-
R16G16B16_SSCALED	FLOAT	R	G	B	-
R16G16B16_USCALED	FLOAT	R	G	B	-
R16G16B16_UINT	UINT	R	G	B	-

Source Element	Converted Component				
Surface Format Name	Format	0	1	2	3
R16G16B16_SINT	SINT	R	G	B	-
R32_SFIXED	FLOAT	R	-	-	-
R10G10B10A2_SNORM	FLOAT	R	G	B	A
R10G10B10A2_USCALED	FLOAT	R	G	B	A
R10G10B10A2_SSCALED	FLOAT	R	G	B	A
R10G10B10A2_SINT	SINT	R	G	B	A
B10G10R10A2_SNORM	FLOAT	R	G	B	A
B10G10R10A2_USCALED	FLOAT	R	G	B	A
B10G10R10A2_SSCALED	FLOAT	R	G	B	A
B10G10R10A2_UINT	UINT	R	G	B	A
B10G10R10A2_SINT	SINT	R	G	B	A
R64G64B64A64_PASSTHRU	NONE	R	G	B	A
R64G64B64_PASSTHRU	NONE	R	G	B	-

DestinationFormatSelection

The **Component Select 0..3** bits are then used to select, on a per-component basis, which destination components will be written and with which value. The supported selections are the converted source component, VertexID, InstanceID, PrimitiveID, the constants 0 or 1.0f, or nothing (VFCOMP_NO_STORE). If a converted component is listed as '-' (not available) in Source Element Formats supported in VF Unit. It must not be selected (via VFCOMP_STORE_SRC), or an UNPREDICTABLE value will be stored in the destination component.

The selection process sequences from component 0 to 3. Once a **Component Select** of VFCOMP_NO_STORE is encountered, all higher-numbered **Component Select** settings must also be programmed as VFCOMP_NO_STORE. It is therefore not permitted to have 'holes' in the destination VE.

Dangling Vertex Removal

The last functional stage of processing of the 3DPRIMITIVE command is the removal of "dangling" vertices. This stage includes the discarding of primitive topologies without enough vertices for a single object (e.g., a TRISTRIP with only two vertices), as well as the discarding of trailing vertices that do not form a complete primitive (e.g., the last two vertices of a 5-vertex TRILIST). 3D APIs typically require these vertices to be (effectively) discarded before the VS stage.

Statistics Gathering

This function is best described as a filter operating on the vertex stream emitted from the processing of the 3DPRIMITIVE. The filter inputs the PrimType, PrimStart, and PrimEnd values associated with the generated vertices. The filter only outputs primitive topologies without dangling vertices. This requires the filter to (a) be able to buffer some number of vertices, and (b) be able to remove dangling vertices from the pipeline and dereference the associated VUE handles.

3DSTATE_VF_STATISTICS

Vertices Generated

VF will increment the IA_VERTICES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) for each vertex it fetches, even if that vertex comes from a cache rather than directly from a vertex buffer in memory. Any “dangling” vertices (fetched vertices that are part of an incomplete object) will not be included.

Objects Generated

VF will increment the IA_PRIMITIVES_COUNT Register (see Memory Interface Registers in vol1a System Overview) for each object (point, line, triangle, or quadrilateral) that it forwards down the pipeline.

Note
For LINELOOP, the last (closing) line object is counted.

Vertex Shader (VS) Stage

The Vertex Shader (VS) stage of the 3D Pipeline is used to perform processing ("shading") of vertices after they are assembled and written to the URB by the VF function. The primary function of the VS stage is to pass vertices that miss in the VS Cache to VS threads, and then pass the VS thread-generated vertices down the pipeline. Vertices that hit in the VS Cache have already been shaded and are therefore passed down the pipeline unmodified.

When the VS stage is disabled, vertices flow through the unit unmodified (i.e., as written by the VF unit).

State

3DSTATE_VS

3DSTATE_CONSTANT_VS

3DSTATE_PUSH_CONSTANT_ALLOC_VS

3DSTATE_BINDING_TABLE_POINTERS_VS

3DSTATE_SAMPLER_STATE_POINTERS_VS

3DSTATE_URB_VS

Functions

This topic is currently under development.

Vertex Shader Cache (VS$)

Note: The VS$ should not be confused with input data caches used by the VF stage when fetching data from index or vertex buffers in memory.

The 3D Pipeline employs a Vertex Shader Cache (VS$) that is shared between the VF and VS stages. (See *Vertex Fetch* chapter for additional information). The vertex index generated by the VF stage is used as the cache tag. The cached data contains the URB handle of a VUE, which in turn typically contains the vertex data output from a previously-executed VS shader, though if the VS function is disabled the VUE will contain the input vertex data generated by the VF stage.

When the VF stage processes a vertex, it will first perform a lookup in the VS$. If the vertex hits in the VS$, the VS stage will return the hit VUE handle to the VF stage, and the VF stage will subsequently pass the returned VUE handle back down the FF pipeline to VS. If the vertex misses in the VS$ (or always, if the VS$ is disabled), the VS stage will allocate a VUE handle for the miss vertex and return this to the VF stage. The VF stage will then proceed to fetch/generate the input vertex data, store the results into the VUE, and then pass the VUE down to the VS stage. If the VS function is enabled, the VUE handle/data will be used as input to a VS shader thread, and that thread will overwrite the VUE with the shader results.

The VS$ may be explicitly DISABLED via the Vertex Cache Disable bit in 3DSTATE_VS. Even when explicitly ENABLED, the VS stage will (by default) implicitly disable the VS$ whenever it detects one of the following conditions:

Condition
Sequential indices are used in the 3DPRIMITIVE command (though this is effectively a don't care as there would not be any VS$ hits).

The implicit disable persists as long as one of these conditions persist, afterwhich the VS$ is invalidated.

The VS$ is implicitly invalidated between 3DPRIMITIVE commands and between instances within a 3DPRIMITIVE command – therefore use of InstanceID in a Vertex Element is not a condition under which the cache is implicitly disabled.

The following table summarizes the modes of operation of the VS$.

VS$	VS Function Enable	Mode of Operation
DISABLED (implicitly or explicitly)	DISABLED	The VS$ is not used. VF stage assembles all vertices and writes them into the VUE supplied by the VS stage. VS stage subsequently passes references to these VUEs down the pipeline without spawning any VS threads. Usage Model: This is an exceptional condition, only required for when the VF-generated vertices contain PrimitiveID. Otherwise the VS$ should be enabled.
	ENABLED	The VS$ is not used. VF stage assembles all vertices and writes them into the VUE supplied by the VS stage. VS stage subsequently spawns VS threads to process all vertices, overwriting the input data with the results. The VS stage pass references to these VUEs down the pipeline. Usage Model: This mode is only used when the VS function is required, but either (a) the VS kernel produces a side effect (e.g., writes to a memory buffer) which in turn requires every vertex to be processed by a VS thread, or (b) the input vertex contains PrimitiveID.
ENABLED	DISABLED	The VS$ is used to provide reuse of VF-generated vertices. The VF stage checks the cache and only processes (assembles/writes) vertices that miss in the VS$. In either case, the VS stage passes references to vertices (that hit or miss) down the pipeline without spawning any VS threads. Usage Model: Normal operation when the VS function is not required (e.g., SW has detected a VS shader that simply copies outputs to inputs).
	ENABLED	The VS$ is used to provide reuse of VS-processed vertices. The VF stage checks the cache and only processes (assembles/writes) vertices that miss in the VS$. The VS stage only processes (shades) the vertices that missed in the VS$. The VS stage sends references to hit or missed vertices down the pipeline in the correct order. Usage Model: Normal operation when the VS function is required and use of the VS$ is permissible.

SIMD4x2 VS Thread Request Generation

Project	Description
CHV, BSW	CHV, BSW adds SIMD8 VS thread request generation, which is described in a following section.

The following discussion assumes the VS Function is ENABLED.

When the Vertex Cache is disabled, the VS unit passes each pair of incoming vertices to a VS thread. Under certain circumstances (e.g., prior to a state change or pipeline flush) the VS unit spawns a VS thread to process a single vertex. Note that, in this case, the "unused" vertex slot is "disabled" via the Execution Mask provided by the VS unit to the GEN4 subsystem as part of the thread dispatch (See the EU ISA volume). The VS thread is itself unaware of the single-vertex case, and therefore a single VS kernel can be used to process one or two vertices. (The performance of single-vertex processing roughly equals the two-vertex case.)

When the Vertex Cache is enabled, the VF unit detects vertices that hit in the cache and marks these vertices so that they bypass VS thread processing and are output via a reference to the cached VUE. The VS unit keeps track of these cache-hit vertices as it proceeds to process cache-miss vertices. The VS unit guarantees that vertices exit the unit in the order they are received. This may require the VS unit to issue single-vertex VS threads to process a cache-miss vertex that has yet to be paired up with another cache-miss vertex (if this condition is preventing the VS unit from producing any output).

SIMD4x2 VS Thread Execution

Project	Description
CHV, BSW	CHV, BSW adds SIMD8 VS thread execution, which is described in a following section.

A VS kernel (with one exception mentioned below) assumes it is to operate on two vertices in parallel. Input data is either passed directly in the thread payload (including the input vertex data) or indirectly via pointers passed in the payload.

Refer to the *EU ISA* chapters for specifics on writing kernels that operate in SIMD4x2 fashion.

Refer to the 3D Pipeline Stage Overview (*3D Overview*) for information on FF-unit/thread interactions.

In the (unlikely) event that the VS kernel needs to determine whether it is processing one or two vertices, the kernel can compare the **URB Return Handle 0** and **URB Return Handle 1** fields of the thread payload. These fields differ if two vertices are being processed, and identical if one vertex is being processed. An example of when this test may be required is if the kernel outputs some vertex-dependent results into a memory buffer; without the test the single vertex case might incorrectly output two sets of results. Note that this is not the case for writing the URB destinations, as the Execution Mask prevents the write of an undefined output.

VS Thread Dispatch Masks

The VS stage controls the initial alue loaded into the EU's Dispatch Mask state register as part of thread dispatch.

SIMD4x2 Dispatch Mask

In SIMD4x2 dispatch mode, the EU Dispatch Mask is initialized as a function of the number of vertices included in the thread dispatch, as follows:

- 1 vertex: 0x0000000F
- 2 vertices: 0x000000FF

SIMD8 Dispatch Mask

In SIMD8 dispatch mode, the EU Dispatch Mask is initialized as a function of the number of vertices included in the thread dispatch, as follows:

- 1 vertex: 0x00000001
- 2 vertices: 0x00000003
- 3 vertices: 0x00000007
- 4 vertices: 0x0000000F
- 5 vertices: 0x0000001F
- 6 vertices: 0x0000003F
- 7 vertices: 0x0000007F
- 8 vertices: 0x000000FF

Vertex Output

VS threads must always write the destination URB entries whose handles are passed in the thread payload. Refer to **Vertex Data Overview** for details on any required contents/formats.

Thread Termination

VS threads must signal thread termination, in all likelihood on the last message output to the URB shared function. Refer to the *ISA* doc for details on End-Of-Thread indication.

Primitive Output

The VS unit will produce an output vertex reference for every input vertex reference received from the VF unit, in the order received. The VS unit simply copies the PrimitiveType, StartPrim, and EndPrim information associated with input vertices to the output vertices, and does not use this information in any way. Neither does the VS unit perform any readback of URB data.

Statistics Gathering

The VS stage tracks a single pipeline statistic, the number of times a vertex shader is executed. A vertex shader is executed for each vertex that is fetched on behalf of a 3DPRIMITIVE command, unless the shaded results for that vertex are already available in the vertex cache. If the **Statistics Enable** bit in VS_STATE is set, the VS_INVOCATION_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) will be incremented for *each vertex* that is dispatched to a VS thread. This counter will often need to be incremented by 2 for each thread invoked since 2 vertices are dispatched to one VS thread in the general case.

Project	Description
CHV, BSW	When **VS Function Enable** is DISABLED and **Statistics Enable** is ENABLED, VS_INVOCATION_COUNT increments by one for every vertex that passes through the VS stage, even though no VS threads are spawned.

Payloads

This topic is currently under development.

SIMD4x2 Payload

The following table describes the payload delivered to VS threads.

VS Thread Payload (SIMD4x2)

DWord	Bits	Description
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset:** Specifies the extent of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID:** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. **Format:** Reserved for HW Implementation Use. **Format:**

DWord	Bits	Description
		<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U9</td></tr></table> **Range:** <table><tr><th>Project</th><th>Range</th></tr><tr><td>CHV, BSW</td><td>0-503</td></tr></table>
	9:0	
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address**or the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). (See *3D Pipeline* for further description). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:16	Reserved
	15:0	**URB Return Handle 1:** This is the 64B-aligned URB offset where the EU's upper channels (DWords 7:4) results are to be stored. If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). (See *Generic FF Unit* for further description). Format: <table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U14 64B-aligned URB offset.</td></tr></table>
R0.0	31:16	Reserved
	15:0	**URB Return Handle 0:** This is the 64B-aligned URB offset where the EU's lower channels

DWord	Bits	Description
		(DWords 3:0) results are to be stored. (See *Generic FF Unit* for further description). Format: **Project** / **Format** CHV, BSW / U14 64B-aligned URB offset.
[Varies] optional	255:0	Constant Data (optional): Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_VS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Varies	255:0	**Vertex Data:** Data from (possibly) one or (more typically) two Vertex URB Entries is passed to the thread in the thread payload. The **Vertex URB Entry Read Offset** and **Vertex URB Entry Read Length** state variables define the regions of the URB entries that are read from the URB and passed in the thread payload. These SVs can be used to provide a subset of the URB data as required by SW. The vertex data is laid out in the thread header in an interleaved format. The lower DWords (0-3) of these GRF registers always contain data from a Vertex URB Entry. The upper DWords (4-7) may contain data from another Vertex URB Entry. This allows two vertices to be processed (shaded) in parallel SIMD8 fashion. The VS kernel is not aware of the validity of the upper vertex.

SIMD8 Payload

The following table describes the payload delivered to VS threads.

SIMD8 VS Thread Payload

DWord	Bits	Description
R0.7	31	
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>(See 3D Pipeline for further description on scratch space allocation).
Format = GeneralStateOffset[31:10]</td></tr>
<tr><td></td><td>9</td><td>Reserved</td></tr>
<tr><td></td><td>8:0</td><td>FFTID: This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit.
Format: Reserved for HW Implementation Use.
Format:
<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U9</td></tr></table>Range:
<table><tr><th>Project</th><th>Range</th></tr><tr><td>CHV, BSW</td><td>0-503</td></tr></table></td></tr>
<tr><td></td><td>9:0</td><td></td></tr>
<tr><td>R0.4</td><td>31:5</td><td>Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address.
Format = SurfaceStateOffset[31:5]</td></tr>
<tr><td></td><td>4:0</td><td>Reserved</td></tr>
<tr><td>R0.3</td><td>31:5</td><td>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.
Format = DynamicStateOffset[31:5] [CHV, BSW]</td></tr>
<tr><td></td><td>4</td><td><table><tr><th>Project</th><th>Description</th></tr><tr><td>CHV, BSW</td><td>Reserved.</td></tr></table></td></tr>
<tr><td></td><td>3:0</td><td>Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two is raised to (over determine the amount of scratch space).
(See 3D Pipeline for further description.)
Format = U4 power of two (in excess of 10)
Range = [0,11] indicating [1K Bytes, 2M Bytes]</td></tr>
<tr><td>R0.2 : R0.0</td><td>31:0</td><td>Reserved: MBZ</td></tr>
<tr><td>R1.7</td><td>31:0</td><td>Vertex 7 URB Return Handle (see R1.0)</td></tr>
<tr><td>R1.6</td><td>31:0</td><td>Vertex 6 URB Return Handle (see R1.0)</td></tr>
</table>

DWord	Bits	Description
R1.5	31:0	Vertex 5 URB Return Handle (see R1.0)
R1.4	31:0	Vertex 4 URB Return Handle (see R1.0)
R1.3	31:0	Vertex 3 URB Return Handle (see R1.0)
R1.2	31:0	Vertex 2 URB Return Handle (see R1.0)
R1.1	31:0	Vertex 1 URB Return Handle (see R1.0)
R1.0	31:16	Reserved
	15:0	**Vertex 0 URB Return Handle.** This is the offset within the URB where Vertex 0 is to be stored. Format: 64B-granular offset into the URB
[Varies] optional	255:0	Constant Data (optional): Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_VS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
		Vertex Data: Input data for the 8 input vertices is located here. Vertex0 data is passed in DW0 of these GRFs, and Vertex 7 data is passed in DW7. The first GRF contains Element 0 Component 0 for all 8 vertices, followed by components 1-3 in the three subsequent GRFs. This is followed by GRFs containing Element 1, and so on, up to the number of elements specified by Vertex URB Read Length. Note that the maximum limit is 30 elements per vertex, though the practical limiit imposed by the compiler is likely lower.
Rv.7	31:0	Vertex 7 Element 0 Component 0
Rv.6	31:0	Vertex 6 Element 0 Component 0
Rv.5	31:0	Vertex 5 Element 0 Component 0
Rv.4	31:0	Vertex 4 Element 0 Component 0
Rv.3	31:0	Vertex 3 Element 0 Component 0
Rv.2	31:0	Vertex 2 Element 0 Component 0
Rv.1	31:0	Vertex 1 Element 0 Component 0
Rv.0	31:0	Vertex 0 Element 0 Component 0
Rv+1.7	31:0	Vertex 7 Element 0 Component 1
Rv+1.6	31:0	Vertex 6 Element 0 Component 1
Rv+1.5	31:0	Vertex 5 Element 0 Component 1
Rv+1.4	31:0	Vertex 4 Element 0 Component 1
Rv+1.3	31:0	Vertex 3 Element 0 Component 1
Rv+1.2	31:0	Vertex 2 Element 0 Component 1
Rv+1.1	31:0	Vertex 1 Element 0 Component 1

DWord	Bits	Description
Rv+1.0	31:0	Vertex 0 Element 0 Component 1
..		Vertex 0-7 Element 0 Component 2,3
..		Vertex 0-7 Element 1 Component 0-3
..		Vertex 0-7 Element 2-N Component 0-3

Hull Shader (HS) Stage

The Hull Shader (HS) stage of the pipeline is used to process patchlist (PATCHLIST_*n*) topologies in support of higher-order surface (HOS) tessellaton. If the HS stage is enabled, each incoming patch object is processed by a possible series of HS threads. The combined output of these threads is a Patch URB Entry ("patch record") written to the URB. This patch record is used by subsequent stages (TE, DS) to complete the HOS tessellation operations.

Programming Note	
Context:	Hull Shader (HS) Stage
For SW Tessellation mode, the HS thread can also write tessellated domain point topologies to memory. The domain point count and starting memory address of the domain points are passed via the Patch Header in the patch record.	

The vertices associated with patchlist primitives are also referred to as "Input Control Points" (ICPs) to contrast them with any "Output Control Points" the HS threads may write to the patch record. (The definition and use of OCPs are outside the scope of this document).

The HS stage also performs statistics counting. Incomplete topologies do not reach the HS stage.

The HS, TE, and DS stages must be enabled and disabled together. When these stages are disabled, all topologies (including patchlist topologies) simply pass through to the GS stage. When these stages are enabled, only patchlist topologies should be issued to the pipeline, otherwise behavior is UNDEFINED.

Programming Note	
Context:	Hull Shader (HS) Stage
Tessellation is not supported in HPCXTs in which case the HS, TE, and DS stages must all be disabled.	

State

This section contains the state registers for the Hull Shader.

3DSTATE_HS

3DSTATE_PUSH_CONSTANT_ALLOC_HS

3DSTATE_CONSTANT_HS

3DSTATE_CONSTANT(Body)

3DSTATE_BINDING_TABLE_POINTERS_HS

3DSTATE_SAMPLER_STATE_POINTERS_HS

3DSTATE_URB_HS

Functions

This topic is currently under development.

Patch Object Staging

The HS unit accepts patchlist topologies as a stream of incoming vertices. Depending on the number of vertices per patch object (as specified by the PATCHLIST_*n* topology), the HS thread assembles each complete patch object and passes it (its vertices, PrimitiveID, etc.) to HS thread(s) as described below.

HS Thread Execution

Input to HS threads is comprised of:

- Input Control Points (incoming patch vertices), pushed into the payload and/or passed indirectly via URB handles.
- Push Constants (common to all threads)
- Patch Data handle
- Resources available via binding table entries (accessed through shared functions)
- Miscellaneous payload fields (Instance Number, etc.)

Typically the only output of the HS threads is the Patch URB Entry (patch record). All thread instances for an input patch are passed the same patch record handle. As the (possibly concurrent) threads can both read and write the patch record, it is up to the kernels to ensure deterministic results. One approach would be to use the thread's Instance Number as an index for URB write destinations.

HS Thread Dispatch Mask

The HS stage controls the initial value loaded into the EU's Dispatch Mask state register as part of thread dispatch.

SINGLE_PATCH Dispatch Mask
In SINGLE_PATCH mode, the EU Dispatch Mask is initialized at thread dispatch to 0x000000FF.

Patch URB Entry (Patch Record) Output

For each patch, the HS thread(s) generate a single patch record, starting with a fixed 32B Patch Header.

When the final thread instance terminates, the patch record handle is passed down the pipeline to the Tessellation Engine (TE).

Please Provide Title

Patch Header DW0-7

The first 8 DWords of the patch record is defined as a "Patch Header". The Patch Header is written by an HS thread and read by the TE stage. It normally contains up to six **Tessellation Factors** (TFs) that determine how finely the TE stage needs to tessellate a domain (if at all).

In SW Tessellation mode, the header contains **Domain Point Count** and **Domain Point Buffer Starting Address** fields, which identify the domain points generated by an HS thread.

The following table shows the fixed layouts of the Patch Header DW0-7, depending on the SW Tessellation Mode. Also see the fixed layouts depending on DomainType that follow.

Patch Header (SW Tessellation Mode)

DWord	Bits	Description
7	31:0	**Domain Point Count** Specifies the number of DOMAIN_POINT structures in the domain point list in memory. If 0, there are no domain points defined, the patch will considered "culled", and the TE stage will discard the patch. Otherwise the TS stage will send this number of domain points down the pipeline. Format: U32
6	31:6	**Domain Point Buffer Starting Address (DPBSA)** This field specifies the starting memory offset from SW Tessellation Base Address (set by the SWTESS_BASE_ADDRESS command) at which the HS thread has written a list of DOMAIN_POINT structures. This field is ignored if **Domain Point Count** is 0. Format: 64B-aligned offset from SW Tessellation Base Address
	5:0	Reserved: MBZ
5-0	31:0	Reserved: MBZ

The following tables show the fixed layouts of the Patch Header DW0-7, depending on DomainType.

Patch Header (QUAD Domain)

DWord	Bits	Description
7	31:0	**UEQ0 Tessellation Factor** Format: FLOAT32
6	31:0	**VEQ0 Tessellation Factor** Format: FLOAT32
5	31:0	**UEQ1 Tessellation Factor** Format: FLOAT32
4	31:0	**VEQ1 Tessellation Factor** Format: FLOAT32
3	31:0	**Inside U Tessellation Factor** Format: FLOAT32
2	31:0	**Inside V Tessellation Factor** Format: FLOAT32

DWord	Bits	Description
1	31:0	Reserved : MBZ
0	31:1	Reserved : MBZ
	0	**Description** **TR DS Cache Disable** Setting this bit will cause the DS Cache to be disabled for tessellator-output topologies resulting from the tessellation of stitch-transition regions. Note that the DS Cache will remain enabled for other topologies generated for the patch (i.e., the inner region if one exists, or the entire tessellated patch if there are no transition regions). The HS kernel shall clear this bit when the patch output topology is POINT. While disabling the DS Cache usually results in reduced performance, when domain points hit in this cache more than 11 times per patch a more significant performance penalty may be incurred. Therefore it is recommended that the HS kernel include an algorithm to set this bit when such conditions are met. A more conservative algorithm (i.e., setting the bit when conditions are close to being met) can be employed in order to limit the execution overhead of this algorithm. Such an algorithm could possibly examine the ratio of each outer edge's tessellation factor to the corresponding inner tessellation factor, thus determining the (possibly-approximated) count of triangles including any given domain point in that transition region. If any of the transition region domain points are included in more than 11 triangles (or may approach that number), the bit should be set for the patch. Note that some domain points (such as the corners of the inside region) can be connected to two adjacent transition regions, therefore cache hits from both regions must be considered. Other exceptional cases include when the inner region of patch with EVEN or INTEGER partitioning collapses to a single point, in which case that central domain point is included in every tessellated triangle.

Patch Header (TRI Domain)

DWord	Bits	Description
7	31:0	**UEQ0 Tessellation Factor** Format: FLOAT32
6	31:0	**VEQ0 Tessellation Factor** Format: FLOAT32
5	31:0	**WEQ0 Tessellation Factor** Format: FLOAT32
4	31:0	**Inside Tessellation Factor** Format: FLOAT32
3-1	31:0	Reserved : MBZ
0	31:1	Reserved : MBZ
	0	**Description** **TR DS Cache Disable** See description of the corresponding bit in the QUAD domain patch header (above).

Patch Header (ISOLINE Domain)

DWord	Bits	Description
7	31:0	**Line Detail Tessellation Factor** Format: FLOAT32
6	31:0	**Line Density Tessellation Factor** Format: FLOAT32
5-0	31:0	Reserved : MBZ

HW Bug: The Tessellation stage will incorrectly add domain points along patch edges under the following conditions, which may result in conformance failures and/or cracking artifacts:

- QUAD domain
- INTEGER partitioning
- All three TessFactors in a given U or V direction (e.g., V direction: UEQ0, InsideV, UEQ1) are all exactly 1.0
- All three TessFactors in the other direction are > 1.0 and all round up to the same integer vaule (e.g, U direction: VEQ0 = 3.1, InsideU = 3.7, VEQ1 = 3.4)

DOMAIN_POINT Structure

In SW Tessellation Mode (i.e., when the TE State is SW_TESS), the TE stage reads a sequence of DOMAIN_POINT structures from memory, starting at the Domain Point Buffer Starting Address field of the patch header. (The DPBSA is treated as an offset from the SW Tessellation Base Address as set by the SWTESS_BASE_ADDRESS command.)

DOMAIN_POINT Memory Structure (SW Tessellation)

DWord	Bits	Description
0	31	**PrimStart** Set on the first domain point of the topology (e.g., first vertex in a TRISTRIP).
	30	**PrimEnd** Set on the last domain point of the topology (e.g., last vertex in a TRISTRIP). Programming note: Software must ensure that incomplete primitives are not output, or behavior is UNDEFINED.
	29	**PatchEnd** Set on the last domain point for the *patch*. By definition, PrimEnd must also be set. **Programming Note:** Software must ensure that the *Domain Point Count* coincides with the domain point marked with PatchEnd.
	28:24	**PrimType** This is the primitive topology type. Format: See 3DPRIMITIVE for encodings Valid values:POINTLIST, LINESTRIP, LINELIST, TRISTRIP, TRISTRIP_REV, TRILIST, TRIFAN.
	23:19	**Reserved**
	18:17	**DS Tag [16:15]**

DWord	Bits	Description
		This field provides bits [16:15] of the DS Tag value for this domain point. See **DS Tag [14:0]** Format: U2
	16:0	**U Coordinate** Format: U1.16
1	31:17	**DS Tag [14:0]** This field provides bits [14:0] of the DS Tag value for this domain point. In order to utilize the DS cache, the 17-bit DS Tag must be unique for the associated U,V coordinate. If software cannot guarantee this, the DS cache must be disabled when in SW Tessellation mode. Format: U15
	16:0	**V Coordinate** Format: U1.16

Statistics Gathering

HS Invocations

The HS unit controls the HS_INVOCATIONS counter, which counts the number of patches processed by the HS stage.

Payloads

This topic is currently under development.

SINGLE_PATCH Payload

The following table shows the layout of the payload delivered to HS threads. Refer to 3D Pipeline Stage Overview (*3D Pipeline*) for details on those fields that are common amongst the various pipeline stages.

Patch object vertex (ICP) data can be passed by value (data pushed in the payload) and/or by reference (URB handle pushed in the payload).

SINGLE_PATCH HS Thread Payload

GRF DWord	Bits	Description
R0.7	31	
	30:0	Reserved.
R0.6	31	Dereference Thread This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.

<table>
<tr><th>GRF DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td>23:0</td><td>Thread ID. This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time.

Format: Reserved for HW Implementation Use.</td></tr>
<tr><td>R0.5</td><td>31:10</td><td>Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address.

Format = GeneralStateOffset[31:10]</td></tr>
<tr><td></td><td>9.0</td><td>Reserved.</td></tr>
<tr><td></td><td>8:0</td><td>FFTID. This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion.

Format:
<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U9</td></tr></table>
Range:
<table><tr><th>Project</th><th>Range</th></tr><tr><td>CHV, BSW</td><td>0-503</td></tr></table></td></tr>
<tr><td>R0.4</td><td>31:5</td><td>Binding Table Pointer: Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address.

Format = SurfaceStateOffset[31:5]</td></tr>
<tr><td></td><td>4:0</td><td>Reserved.</td></tr>
<tr><td>R0.3</td><td>31:5</td><td>Sampler State Pointer. Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address.

Format = DynamicStateOffset[31:5]</td></tr>
<tr><td></td><td>4</td><td>Reserved.</td></tr>
<tr><td></td><td>3:0</td><td>Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).

Programming Notes:

This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it.

Format = U4 power of two (in excess of 10)</td></tr>
</table>

<table>
<tr><th>GRF DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>Range = [0,11] indicating [1K Bytes, 2M Bytes]</td></tr>
<tr><td rowspan="4">R0.2</td><td>31:24</td><td>Reserved.</td></tr>
<tr><td>23:17</td><td>Instance Number. A patch-relative instance number between 0 and InstanceCount-1.
Format = U7</td></tr>
<tr><td>16:13</td><td>Barrier Index. This index is to be used in any BarrierMsgs sent by this thread to the Gateway.
Format = U4</td></tr>
<tr><td>12:0</td><td>Reserved.</td></tr>
<tr><td>R0.1</td><td>31:0</td><td>Primitive ID. This field contains the Primitive ID associated with the patch.
Format: U32</td></tr>
<tr><td rowspan="2">R0.0</td><td>31:16</td><td>Reserved.</td></tr>
<tr><td>15:0</td><td>Patch Data Record URB Return Handle.
Format:<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U14 64B-aligned URB offset.</td></tr></table></td></tr>
<tr><td colspan="3">R1 is only included for dispatches that have Include Vertex Handles enabled.</td></tr>
<tr><td rowspan="2">R1.7</td><td>31:16</td><td>ICP 7 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 7 Handle
Format:<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U14 64B-aligned URB offset.</td></tr></table></td></tr>
<tr><td rowspan="2">R1.6</td><td>31:16</td><td>ICP 6 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 6 Handle</td></tr>
<tr><td rowspan="2">R1.5</td><td>31:16</td><td>ICP 5 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 5 Handle</td></tr>
<tr><td rowspan="2">R1.4</td><td>31:16</td><td>ICP 4 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 4 Handle</td></tr>
<tr><td rowspan="2">R1.3</td><td>31:16</td><td>ICP 3 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 3 Handle</td></tr>
<tr><td rowspan="2">R1.2</td><td>31:16</td><td>ICP 2 Handle ID</td></tr>
<tr><td>15:0</td><td>ICP 2 Handle</td></tr>
<tr><td>R1.1</td><td>31:16</td><td>ICP 1 Handle ID</td></tr>
</table>

GRF DWord	Bits	Description
	15:0	ICP 1 Handle
R1.0	31:16	ICP 0 Handle ID
	15:0	ICP 0 Handle
R2 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >7		
R2.7	31:16	ICP 15 Handle ID
	15:0	ICP 15 Handle
R2.6	31:16	ICP 14 Handle ID
	15:0	ICP 14 Handle
R2.5	31:16	ICP 13 Handle ID
	15:0	ICP 13 Handle
R2.4	31:16	ICP 12 Handle ID
	15:0	ICP 12 Handle
R2.3	31:16	ICP 11 Handle ID
	15:0	ICP 11 Handle
R2.2	31:16	ICP 10 Handle ID
	15:0	ICP 10 Handle
R2.1	31:16	ICP 9 Handle ID
	15:0	ICP 9 Handle
R2.0	31:16	ICP 8 Handle ID
	15:0	ICP 8 Handle
R3 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >15		
R3.7	31:16	ICP 23 Handle ID
	15:0	ICP 23 Handle
R3.6	31:16	ICP 22 Handle ID
	15:0	ICP 22 Handle
R3.5	31:16	ICP 21 Handle ID
	15:0	ICP 21 Handle
R3.4	31:16	ICP 20 Handle ID
	15:0	ICP 20 Handle
R3.3	31:16	ICP 19 Handle ID
	15:0	ICP 19 Handle
R3.2	31:16	ICP 18 Handle ID
	15:0	ICP 18 Handle
R3.1	31:16	ICP 17 Handle ID
	15:0	ICP 17 Handle
R3.0	31:16	ICP 16 Handle ID

GRF DWord	Bits	Description
	15:0	ICP 16 Handle
R4 is only included for dispatches that have Include Vertex Handles enabled and when ICP Count >23		
R4.7	31:16	ICP 31 Handle ID
	15:0	ICP 31 Handle
R4.6	31:16	ICP 30 Handle ID
	15:0	ICP 30 Handle
R4.5	31:16	ICP 29 Handle ID
	15:0	ICP 29 Handle
R4.4	31:16	ICP 28 Handle ID
	15:0	ICP 28 Handle
R4.3	31:16	ICP 27 Handle ID
	15:0	ICP 27 Handle
R4.2	31:16	ICP 26 Handle ID
	15:0	ICP 26 Handle
R4.1	31:16	ICP 25 Handle ID
	15:0	ICP 25 Handle
R4.0	31:16	ICP 24 Handle ID
	15:0	ICP 24 Handle
[Varies] optional	255:0	Constant Data (optional): Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_HS command (taking the buffer enables into account).
[Varies] optional	255:0	ICP Vertex Data (optional): There can be up to 32 vertices supplied, each with a size defined by the Vertex URB Entry Read Length state. Vertex 0 DWord 0 is located at Rn.0, Vertex 0 DWord 1 is located at Rn.1, etc. Vertex 1 DWord 0 immediately follows the last DWord of Vertex 0, and so on.

Tessellation Engine (TE) Stage

When enabled, the Tessellation Engine (TE) stage performs fixed-function domain tessellation (decomposition into smaller objects) of incoming patches, as referenced by an HS-generated input PDR handle and as controlled by TE state and Tessellation Factors (TFs) read from the Patch URB Entry (patch record). The TE stage is entirely fixed-function and does not spawn threads.

Description
The TE stage can also operate in SW Tessellation mode, where it simply reads "pre-tessellated" domain point topologies from memory and passes them down the pipeline.

The fixed-function tessellation algorithm is considered an implementation detail and is therefore beyond the scope of this document. That detail includes both the order of output topologies as well as the order of vertices (domain points) within the output topologies. Only a high-level overview is provided to describe how the (few) state variables can be used to control aspects of tessellation behavior. The implementation will generate deterministic results (given the same exact inputs it will produce exactly the same outputs).

Several domain types (QUAD, TRI, and ISOLINE) are supported. Depending on the domain type, the TE stage outputs the required point/line/triangle topologies including a domain point per vertex. These topologies will be output to the DS stage, where the domain points will be converted to 3D object vertices, resulting in 3D objects as typically input to the 3D pipeline when HOS tessellation is not used.

The HS, TE, and DS stages must be enabled and disabled together. When these stages are disabled, all topologies (including patchlist topologies) simply pass through to the GS stage. When these stages are enabled, only patchlist topologies should be issued to the pipeline, else behavior is UNDEFINED. The MI_TOPOLOGY_FILTER command can be used to ensure this happens, i.e., it can be used to have the Command Stream ignore 3DPRIMITIVE commands that do not match a specific topology type.

State

This section contains the state registers for the Tessellation Engine.

3DSTATE_TE

Functions

This topic is currently under development.

Patch Culling

Normally, if any "outside" TF is <= 0.0 or NaN, the entire patch is culled at the TE stage.

Inside TFs are not used to cull patches.

Tessellation Factor Limits

After the Patch Culling test is performed, the TessFactors undergo a min() clamp to either the **MaxTessFactorOdd** (for FRACTIONAL_ODD partitioning) or **MaxTessFactorNotOdd** (for FRACTIONAL_EVEN or INTEGER partitioning). Exception: If the ISOLINE domain is specified, the LineDensity TessFactor will be clamped to the **MaxFactorNotOdd** value even if FRACTIONAL_ODD partitioning is specified).

Usage Note: These max TessFactor values shall be programmed to values required by the APIs (refer to the 3DSTATE_TE definition).

Partitioning

The Partitioning state controls how the TFs are used to divide their corresponding edges.

- INTEGER: The edge is divided into an integral number of equal segments (given some fixed-point tolerance).

  After clamping, the TF is rounded up to an integer value. The edge is divided into that many equal segments.

- EVEN_FRACTIONAL: The edge is divided into an *even* number of possibly-unequal segments. The total number of segments is determined by rounding up the post-clamped TF to an even number.

  More specifically, the edge is divided exactly in half. Like the endpoints of the edge, the midpoint of the edge is by definition a tessellation point. Each half contains some number of equal segments and possibly one smaller segment. The size of the smaller segment is determined by the position of the TF value within the range defined by the TF rounded down and up to even numbers. The closer the TF is to the smaller value, the smaller the segment size is. When the TF reaches the smaller even value, the smaller segment disappears. The closer the TF gets to the larger even value, the closer the smaller segment size approaches the size of the other segments. When the TF reaches the larger even value, all segments are equal. The position of the smaller segment along the half edge varies as a function of the TF value.

- ODD_FRACTIONAL: The edge is divided into an *odd* number of possibly-unequal segments. The tessellation scheme is very similar to EVEN_FRACTIONAL partitioning, except that the edge midpoint is not included as a tessellation point. This, and the fact that the tessellation points are mirrored about the edge midpoint, causes an "odd" segment (which may or may not be the "smaller" segment) to straddle the edge midpoint, therefore resulting in the number of segments for the edge always being odd.

Domain Types and Output Topologies

The major (if only) task of the TE stage is to tessellate a 2D (u,v) domain region, as selected by the Domain state, into some number of 2D object topologies. (If the patch is culled, that number may be zero). The options for Domain state are:

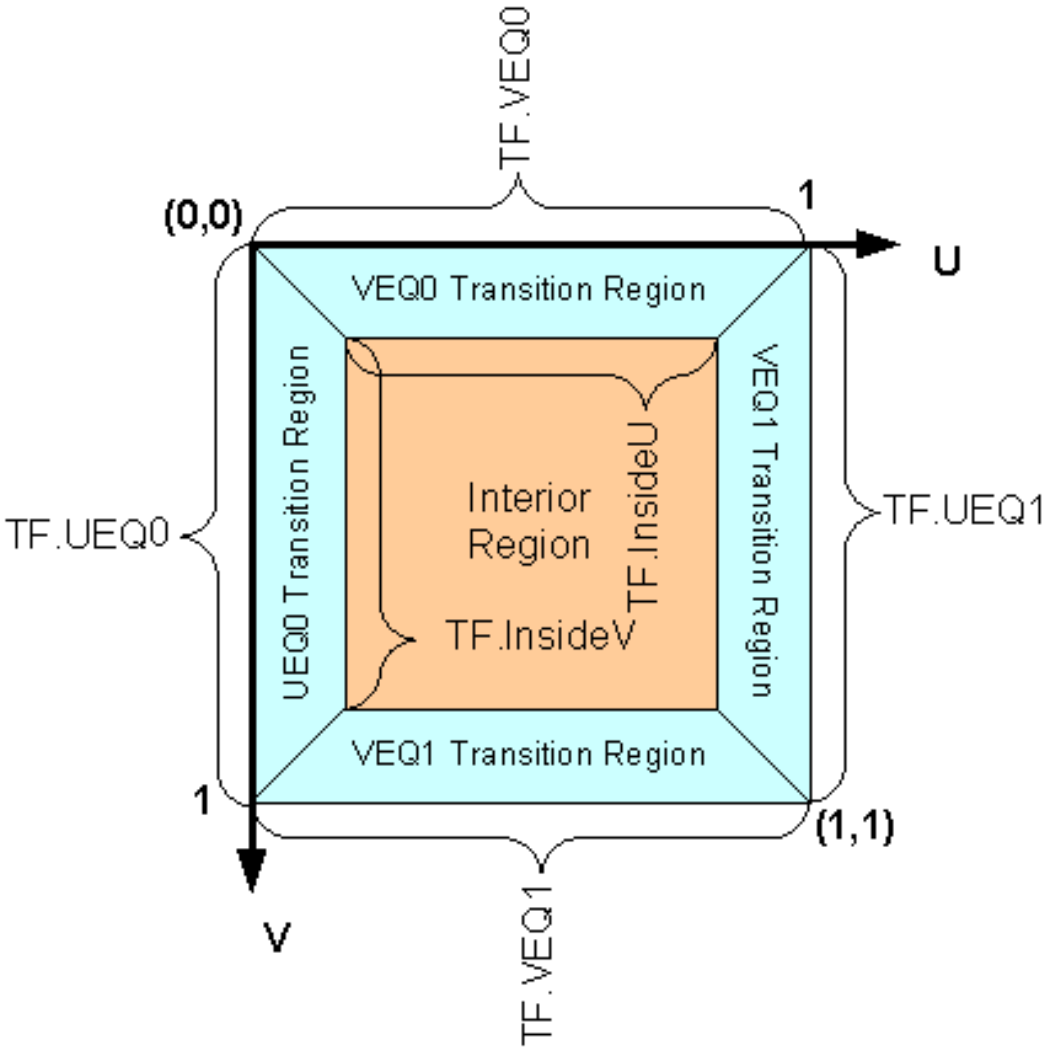
- QUAD: A square 2D region within a u,v Cartesian (rectanguar) space. The region extends from the origin to u=1 and v=1. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- TRI: A triangular 2D region with u,v,w barycentric (areal) coordinates. The three edges correspond to u=0, v=0, and w=0 boundaries. In barycentric coordinates, w = 1 – u – v, therefore points within the region are fully defined as 2D (u,v) coordinates. Within the region, tessellation domain locations are determined. The possible output topologies include points, clockwise triangles, and counter-clockwise triangles.
- ISOLINE: A series of points within a QUAD domain, where the points lie on lines parallel to the u axis and extending from [0,1) in the v direction. Either the segmented lines (linestrips) or individual point topologies can be output.

QUAD Domain Tessellation

The four “outside” TFs (TF.UEQ0, TF.VEQ0, TF.UEQ1, TF.VEQ1) are used to specify the level of tessellation along the four corresponding edges of the 2D quad domain. The two “inside” TFs (TF.InsideU, TF.InsideV) are used to determine the level of tessellation within a 2D “interior” region. Typically the interior region appears as a “regularly-tessellated 2D grid”, however under certain conditions the interior region may collapse in which case only the outside TFs are relevant.

In general, a transition region exists between each edge of the interior region and the corresponding outside edge. The topologies generated for these regions effectively “stitch together” locations along the outside and inside edges, as each of these edges can contain a different number of tessellated segments. In the case where all TFs in a given direction (e.g., TF.VEQ0, TF.InsideU, and TF.VEQ1) are the same value, it appears as if the regularly-tessellated interior region extends all the way to the outside edges. If this condition simultaneously exists for both u and v directions, the entire domain will appear to be tessellated into a regular grid, with no noticeable transition regions.

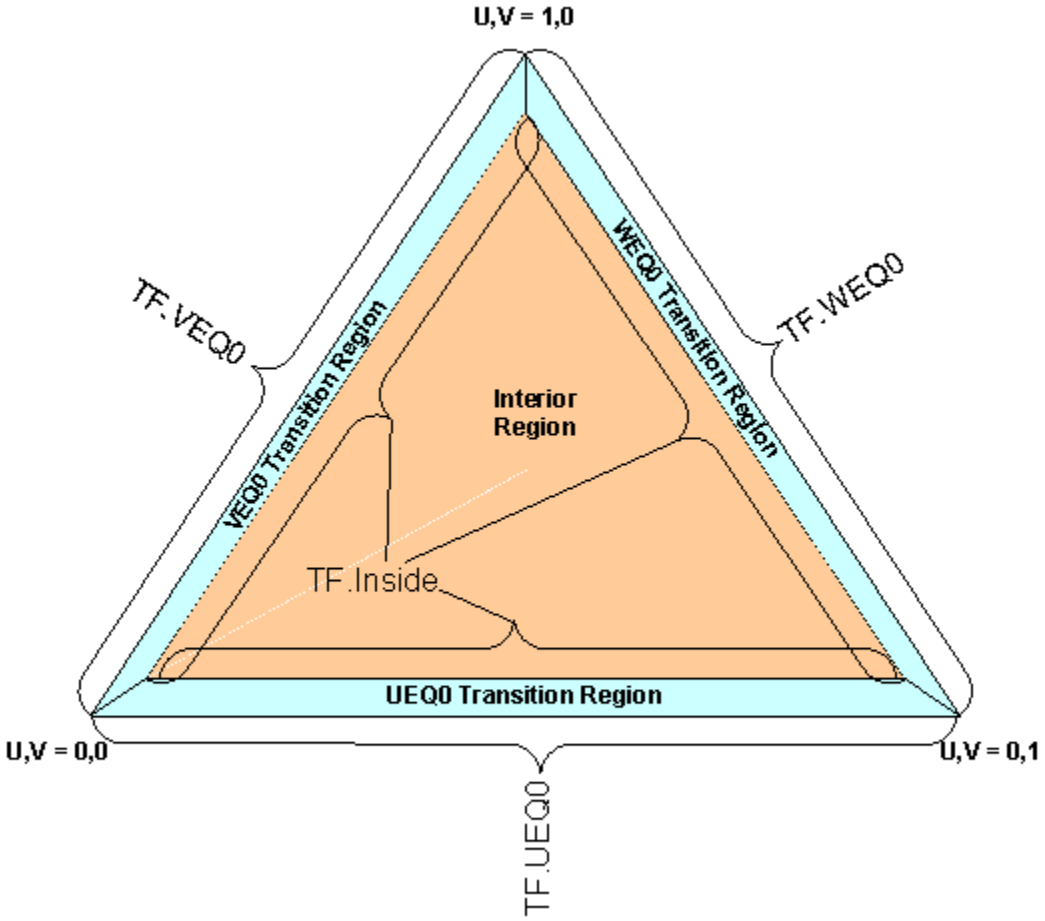
QUAD Domain


TF.VEQ0
(0,0)
1
U
VEQ0 Transition Region
UEQ0 Transition Region
VEQ1 Transition Region
Interior Region
TF.InsideU
TF.InsideV
TF.UEQ0
TF.UEQ1
1
V
(1,1)
TF.VEQ1

TRI Domain Tessellation

Tessellation of the TRI domain is similar to the QUAD domain, except only three outside edges/TFs are used, and the tessellation of the interior region is controlled by a single TF.

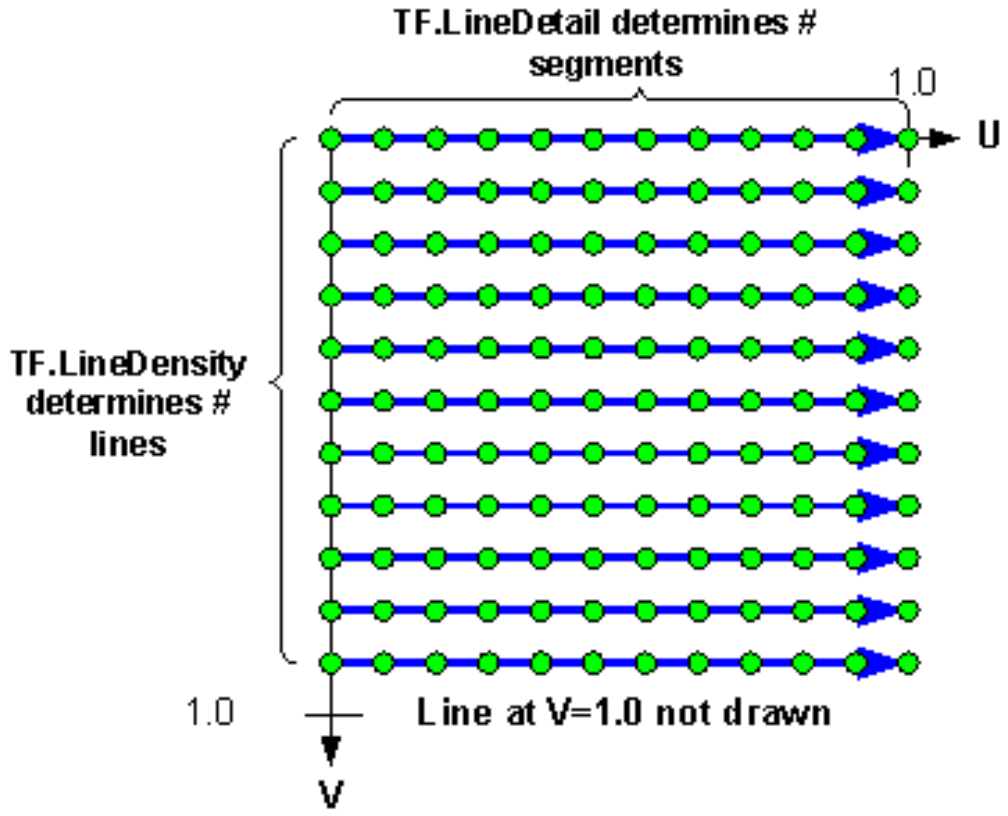
TRI Domain



ISOLINE Domain Tessellation

Tessellation of the ISOLINE domain is different but much simpler than QUAD and TRI domains. The TF.LineDetail TF controls how finely the U direction is tessellated, while the TF.LineDensity TF controls how finely the V direction is tessellated. When LINE output topology is selected, a series of segmented lines parallel to the U axis (constant V) are output. When POINT output topology is selected, only the line segment endpoints are output (as point objects). In either case there is no topology output for the V=1 edge, which avoids overlapping lines for adjacent patches.

ISOLINE Domain



Domain Shader (DS) Stage

The DS stage is very similar to the VS stage in that it is responsible for dispatching EU threads to shade vertices and maintaining a cache (with reference counts) of the shaded vertex outputs of these threads. Major differences are as follows:

- The DS receives topologies with “domain points” instead of vertices. The only data specific to a domain point are its U,V coordinates. These coordinates (plus a default or computed W coordinate) are passed directly in the DS thread payload. There is no other vertex-specific “input vertex data”.
- The concatenation of the domain point U,V coordinates (vs. a vertex index) is used as the cache tag.
- The cache is invalidated between patches.

The DS stage accepts state information via the inline 3DSTATE_DS command.

State

This section contains the state registers for the Domain Shader.

3DSTATE_DS

3DSTATE_PUSH_CONSTANT_ALLOC_DS

3DSTATE_CONSTANT_DS

3DSTATE_CONSTANT(Body)

3DSTATE_BINDING_TABLE_POINTERS_DS

3DSTATE_SAMPLER_STATE_POINTERS_DS

3DSTATE_URB_DS

Functions

This topic is currently under development.

SIMD4x2 Thread Execution

Project	Description
CHV, BSW	A DS kernel assumes it is to operate on up to eight domain points in parallel using the EU's SIMD8 execution model, or on two domain points in parallel (using the EU’s SIMD4x2 execution model).

Refer to the ISA chapters for specifics on writing kernels that operate in SIMD4x2 fashion.

DS threads must always write the destination URB handles passed in the payload. DS threads are not permitted to request additional destination handles. Refer to 3D Pipeline Stage Overview (*3D Overview*) for details on how destination vertices are written and any required contents/formats.

DS threads must signal thread termination on the last message output to the URB shared function.

Statistics Gathering

The DS stage maintains the DS_INVOCATIONS statistics counter, which counts the number of incoming domain points, irrespective of cache hit/miss. Note that this is different than VS_INVOCATIONS, which counts shader invocations and therefore doesn't count cache hits.

Payloads

This topic is currently under development.

SIMD4x2 Payload

The following table describes the payload delivered to DS threads.

DS Thread Payload (SIMD4x2)

DWord	Bits	Description
R0.7	31	
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use. [CHV, BSW] Format = U9
	9:0	
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved

DWord	Bits	Description
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the **General State Base Address** or **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:26	Reserved
	25:16	**Description** **Handle ID 1.** This ID is assigned by the FF unit and used to identify the URB Return Handle 1 to the FF unit (as FF-specific index value, not a URB address). If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). Format = Reserved for HW Implementation Use.
	15:14	Reserved
	13:0	**URB Return Handle 1:** This is the URB handle where Vertex 1 data (the EU's upper channels (DWords 7:4)) results are to be stored. If only one vertex is to be processed (shaded) by the thread, this field will effectively be ignored (no results are stored for these channels, as controlled by the thread's Channel Mask). Format: **Project** / **Format** CHV, BSW: U14 handle (512-bit granular)
R0.0	31:26	Reserved
	25:16	**Description** **Handle ID 0.** This ID is assigned by the FF unit and used to identify the URB Return Handle 0 to the FF unit (as FF-specific index value, not a URB address). Format = Reserved for HW Implementation Use.
	15:14	Reserved

DWord	Bits	Description
	13:0	**URB Return Handle 0:** This is the URB handle where Vertex 0 data (the EU's lower channels (DWords 3:0)) results are to be stored. Format: Project: CHV, BSW — Format: U14 handle (512-bit granular)
R1.7	31:0	**PrimitiveID.** This is the 32-bit PrimitiveID value associated with the patch. It is common to all output vertices resulting from the tessellation of the patch. Format: U32
R1.6	31:0	**Domain Point 1 W Coordinate.** (See Domain Point 0 W Coordinate) Format: FLOAT32
R1.5	31:0	**Domain Point 1 V Coordinate.** (See Domain Point 0 V Coordinate) Format: FLOAT32
R1.4	31:0	**Domain Point 1 U Coordinate.** (See Domain Point 0 U Coordinate) Format: FLOAT32
R1.3	31:14	Reserved
	13:0	**Patch URB Handle.** This is the URB handle of the Patch Record (common to both vertices). Format: Project: CHV, BSW — Format: U14 handle)
R1.2	31:0	**Domain Point 0 W Coordinate.** If **Compute W Coordinate Enable** is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R1.1	31:0	**Domain Point 0 V Coordinate.** V coordinate associated with Domain Point 0. Format: FLOAT32
R1.0	31:0	**Domain Point 0 U Coordinate.** U coordinate associated with Domain Point 0. Format: FLOAT32
Varies [Optional]	255:0	**Constant Data (optional).** Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided

DWord	Bits	Description
		is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_DS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Varies [Optional]	255:0	**Patch URB Data (optional).** Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS). The Patch Data arrives in a non-interleaved format.

SIMD8 Payload

The following table describes the payload delivered to DS threads.

DS Thread Payload (SIMD8)

DWord	Bits	Description
R0.7	31	
	30:0	Reserved
R0.6	31:24	Reserved
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Offset.** Specifies the offset of the scratch space allocated to the thread, specified as a 1KB-granular offset from the **General State Base Address**. See **Scratch Space Base Offset** description in VS_STATE. (See *3D Pipeline* for further description on scratch space allocation). Format = GeneralStateOffset[31:10]
	9	Reserved
	8:0	**FFTID.** This ID is assigned by the FF unit and used to identify the thread within the set of outstanding threads spawned by the FF unit. Format: Reserved for HW Implementation Use. [CHV, BSW] **Format:** U9
	9:0	
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address. Format = SurfaceStateOffset[31:5]
	4:0	Reserved

DWord	Bits	Description
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table to be used by this thread, specified as a 32-byte granular offset from the General State Base Address or Dynamic State Base Address. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space). Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:0	Reserved: delivered as zeros (reserved for message header fields)
R0.1	31:0	**PrimitiveID.** This is the 32-bit PrimitiveID value associated with the patch. It is common to all output domain points resulting from the tessellation of the patch. Format: U32
R0.0	31:27	Reserved
	26:16	**Patch Handle ID.** This ID is assigned by the FF unit and used to identify the patch URB entry to the FF unit (as FF-specific index value, not a URB address). Format = Reserved for HW Implementation Use.
	15:0	**Patch URB Offset.** This is the offset within the URB where the patch data is stored. Format: U14 64B-granular offset into the URB
R1.7	31:0	**Domain Point 7 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.6	31:0	**Domain Point 6 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.5	31:0	**Domain Point 5 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.4	31:0	**Domain Point 4 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.3	31:0	**Domain Point 3 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.2	31:0	**Domain Point 2 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.1	31:0	**Domain Point 1 U Coordinate.** (See Domain Point 0 U Coordinate.)
R1.0	31:0	**Domain Point 0 U Coordinate.** U coordinate associated with Domain Point 0. Format: FLOAT32
R2.7	31:0	**Domain Point 7 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.6	31:0	**Domain Point 6 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.5	31:0	**Domain Point 5 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.4	31:0	**Domain Point 4 V Coordinate.** (See Domain Point 0 V Coordinate.)

DWord	Bits	Description
R2.3	31:0	**Domain Point 3 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.2	31:0	**Domain Point 2 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.1	31:0	**Domain Point 1 V Coordinate.** (See Domain Point 0 V Coordinate.)
R2.0	31:0	**Domain Point 0 V Coordinate.** V coordinate associated with Domain Point 0. Format: FLOAT32
R3.7	31:0	**Domain Point 7 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.6	31:0	**Domain Point 6 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.5	31:0	**Domain Point 5 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.4	31:0	**Domain Point 4 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.3	31:0	**Domain Point 3 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.2	31:0	**Domain Point 2 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.1	31:0	**Domain Point 1 W Coordinate.** (See Domain Point 0 W Coordinate.)
R3.0	31:0	**Domain Point 0 W Coordinate.** If Compute W Coordinate Enable is set, this field will receive the computed value (1 – U – V) for Domain Point 0. Otherwise it is passed as 0.0. Format: FLOAT32
R4.7	31:0	**Domain Point 7 URB Return Handle.** (See R4.0.)
R4.6	31:0	**Domain Point 6 URB Return Handle.** (See R4.0.)
R4.5	31:0	**Domain Point 5 URB Return Handle.** (See R4.0.)
R4.4	31:0	**Domain Point 4 URB Return Handle.** (See R4.0.)
R4.3	31:0	**Domain Point 3 URB Return Handle.** (See R4.0.)
R4.2	31:0	**Domain Point 2 URB Return Handle.** (See R4.0.)
R4.1	31:0	**Domain Point 1 URB Return Handle.** (See R4.0.)
R4.0	31:16	Reserved
	15:0	**Domain Point 0 URB Return Handle.** This is the offset within the URB where domain point 0 is to be stored. Format: U14 64B-granular offset into the URB
Varies [Optional]	255:0	**Constant Data (optional).** Some amount of constant data (possible none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_DS command (taking the buffer enables into account).
Varies [Optional]	255:0	**Patch URB Data (optional).** Some amount of Patch Data (possible none) can be extracted from the URB and passed to the thread in this location in the payload. The amount of data provided is defined by the Patch URB Entry Read Length state (3DSTATE_DS).

Geometry Shader (GS) Stage

This topic currently under development.

GS Stage Overview

The GS stage of the 3D Pipeline converts objects within incoming primitives into new primitives through use of a spawned thread. When enabled, the GS unit buffers incoming vertices, assembles the vertices of each individual object within the primitives, and passes those object vertices (along with other data) to the graphics subsystem for processing by a GS thread.

When the GS stage is disabled, vertices flow through the unit unmodified.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Pipeline* chapter for a general description of a 3D Pipeline stage, as much of the GS stage operation and control falls under these "common" functions. I.e., most stage state variables and GS thread payload parameters are described in *3D Pipeline*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the GS stage, and any exceptions the GS stage exhibits with respect to common FF unit functions.

State

This sections contains the state registers for the Geometry Shader.

For CHV, BSW and CHV, BSW, the state used by GS is defined with this inline state packet.

3DSTATE_GS

3DSTATE_CONSTANT_GS

3DSTATE_CONSTANT(Body)

3DSTATE_PUSH_CONSTANT_ALLOC_GS

3DSTATE_BINDING_TABLE_POINTERS_GS

3DSTATE_SAMPLER_STATE_POINTERS_GS

3DSTATE_URB_GS

Functions

This topic is currently under development.

Object Staging

The GS unit's Object Staging Buffer (OSB) accepts primitive topologies as a stream of incoming vertices, and spawns a thread for each individual object within the topology.

Thread Request Generation

This topic is currently under development.

Object Vertex Ordering

The following table defines the number and order of object vertices passed in the Vertex Data portion of the GS thread payload, assuming an input topology with *N* vertices. The ObjectType passed to the thread is, by default, the incoming PrimTopologyType. Exceptions to this rule (for the TRISTRIP variants) are called out.

The following table also shows which vertex is selected to provide PrimitiveID (bold, underlined vertex number). In general, the vertex selected is the last vertex for non-adjacent prims, and the next-to-last vertex for adjacent prims. Note, however, that there are exceptions:

- reorder-enabled TRISTRIP[_REV], TRISTRIP_ADJ
- "odd-numbered" objects in TRISTRIP_ADJ

PrimTopologyType	Order of Vertices in Payload	
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
POINTLIST	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
POINTLIST_BF	N/A	
LINELIST (N is multiple of 2)	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
LINELIST_ADJ (N is multiple of 4)	[0] = (0,1,2,3); [1] = (4,5,6,7); ...; [(N/4)-1)] = (N-4,N-3,N-2,N-1)	
LINESTRIP (N >= 2)	[0] = (0,1); [1] = (1,2); ...; [N-2] = (N-2,N-1)	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT (N >= 4)	[0] = (0,1,2,3); [1] = (1,2,3,4); ...; [N-4] = (N-4,N-3,N-2,N-1)	LINESTRIP_ADJ_CONT is added. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINESTRIP_BF	N/A	
LINESTRIP_CONT	Same as LINESTRIP	Handled same as LINESTRIP
LINESTRIP_CONT_BF	Same as LINESTRIP	Handled same as LINESTRIP
LINELOOP (N >= 2)	[0] = (0,1); [1] = (1,2); [N] = (N-1,0);	Not supported after GS.
TRILIST (N is multiple of 3)	[0] = (0,1,2); [1] = (3,4,5); ...; [(N/3)-1] = (N-3,N-2,N-1)	
RECTLIST, RECTLIST_SUBPIXEL	Same as TRILIST	Handled same as TRILIST
TRILIST_ADJ (N is multiple of 6)	[0] = (0,1,2,3,4,5); [1] = (6,7,8,9,10,11); ...; [(N/6)-1] = (N-6,N-5,N-4,N-3,N-2,N-1)	
TRISTRIP (Reorder Leading) (N >= 3)	[0] = (0,1,2); {TRISTRIP} [1] = (1,3,2); {TRISTRIP_REV} [k even] = (k,k+1,k+2) {TRISTRIP} [k odd] = (k,k+2,k+1) {TRISTRIP_REV} [N-3] = (see above)	"Odd" triangles have vertices reordered and identified as TRISTRIP to inform the thread.
TRISTRIP (Reorder Trailing)	[0] = (0,1,2) {TRISTRIP}	"Odd" triangles have vertices reordered and identified as TRISTRIP_REV to inform the thread.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
(N >= 3)	[1] = (2,1,3) {TRISTRIP_REV}; ... [k even] = (k,k+1,k+2) {TRISTRIP} [k odd] = (k+1,k,k+2) {TRISTRIP_REV} [N-3] = (see above)	
TRISTRIP_REV (Reorder Leading) (N >= 3)	[0] = (0,2,1) {TRISTRIP_REV}; [1] = (1,2,3) {TRISTRIP}; ...; [k even] = (k,k+2,k+1) {TRISTRIP_REV} [k odd] = (k,k+1,k+2) {TRISTRIP} [N-3] = (see above)	"Even" triangles have vertices reordered and identified as TRISTRIP to inform the thread.
TRISTRIP_REV (Reorder Trailing) (N >= 3)	[0] = (1,0,2) {TRISTRIP_REV} [1] = (1,2,3) {TRISTRIP}; ...; [k even] = (k+1,k,k+2,) {TRISTRIP_REV} [k odd] = (k,k+1,k+2) {TRISTRIP} [N-3] = (see above)	"Even" triangles have vertices reordered and identified as TRISTRIP_REV to inform the thread.
TRISTRIP_ADJ (Reorder Leading) (N >= 6)	N = 6 or 7: [0] = (0,1,2,5,4,3) N = 8 or 9: [0] = (0,1,2,6,4,3); [1] = (2,5,6,7,4,0); ...; N >= 10: [0] = (0,1,2,6,4,3);	Objects have vertices reordered.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
	[1] = (2,5,6,8,4,0); ...; [k>1, even] = (2k,2k-2, 2k+2, 2k+6,2k+4, 2k+3); [k>2, odd] = (2k, 2k+3, 2k+4, 2k+6, 2k+2, 2k-2);...; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1,N-2,N-3); [(N/2)-3, odd] = (N-6,N-3,N-2,N-1,N-4,N-8);	
TRISTRIP_ADJ (Reorder Trailing) (N >= 6)	N = 6 or 7: [0] = (0,1,2,5,4,3) N = 8 or 9: [0] = (0,1,2,6,4,3); [1] = (4,0,2,5,6,7); ...; N >= 10: [0] = (0,1,2,6,4,3); [1] = (4,0,2,5,6,8); ...; [k>1, even] = (2k,2k-2, 2k+2, 2k+6,2k+4, 2k+3); [k>2, odd] = (2k+2, 2k-2, 2k, 2k+3, 2k+4, 2k+6);...; Trailing object: [(N/2)-3, even] = (N-6,N-8,N-4,N-1,N-2,N-3); [(N/2)-3, odd] = (N-4,N-8,N-6,N-3,N-2,N-1);	OpenGL ordering rules (last non-adjacent vertex is the last – aka provoking – vertex of the triangle). Even triangles have the same ordering as Leading Vertex, odd triangle ordering is different (rotated 2 vertices).
TRIFAN (N > 2)	[0] = (0,1,2); [1] = (0,2,3); ...;	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices in Payload [<object#>] = (<vert#>,...); [{modified PrimType passed to thread}]	GS Notes
	[N-3] = (0, N-2, N-1);	
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON, POLYGON_CONT	Same as TRIFAN	POLYGON_CONT is added. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
	[0] = (0,1,2,3); [1] = (4,5,6,7); ...; [(N/4)-1] = (N-4,N-3,N-2,N-1);	Not supported after GS. [CHV, BSW]: QUADLIST primitives are converted into POLYGONS in VF, and therefore never reach the GS.
	[0] = (0,1,3,2); [1] = (2,3,5,4); ... ; [(N/2)-2] = (N-4,N-3,N-1,N-2);	Not supported after GS. [CHV, BSW]: QUADSTRIP primitives are converted into POLYGONS in VF, and therefore never reach the GS.

Project:	CHV, BSW	
[CHV, BSW]: PATCHLIST_1 PATCHLIST_2 PATCHLIST_3..32	[0] = (0); [1] = (1); ...; [N-2] = (N-2); [0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1) similar to above	

Thread Execution

A GS thread is capable of performing arbitrary algorithms given the thread payload (especially vertex) data and associated data structures (binding tables, sampler state, etc.) as input. Output can take the form of vertices output to the FF pipeline (at the GS unit) and/or data written to memory buffers via the DataPort.

The primary usage models for GS threads include (possible combinations of):

- Compiled application-provided "GS shader" programs, specifying an algorithm to convert the vertices of an input object into some output primitives. For example, a GS shader may convert lines of a line strip into polygons representing a corresponding segment of a blade of grass centered on the line. Or it could use adjacency information to detect silhouette edges of triangles and output polygons extruding out from the those edges. Or it could output absolutely nothing, effectively terminating the pipeline at the GS stage.
- Driver-generated instructions used to write pre-clipped vertices into memory buffers (see Stream Output below). This may be required whether or not an app-provided GS shader is enabled.
- Driver-generated instructions usedto emulate API functions not supported by specialized hardware. These functions might include (but are not limited to):
    - Conversion of API-defined topologies into topologies that can be rendered (e.g., LINELOOP→LINESTRIP, POLYGON→TRIFAN, QUADs→TRIFAN, etc.)
    - Emulation of "Polygon Fill Mode", where incoming polygons can be converted to points, lines (wireframe), or solid objects.
    - Emulation of wide/sprite points.

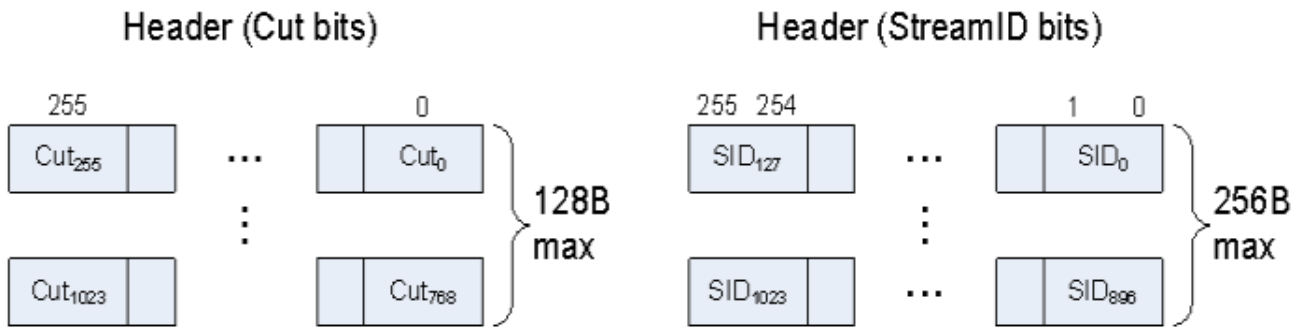
When rendering is required, concurrent GS threads must use the FF_SYNC message (URB shared function) to request an initial VUE handle and synchronize output of VUEs to the pipeline (see *URB* in *Shared Functions*). Only one GS thread can be outputting VUEs to the pipeline at a time. To achieve parallelism, GS threads should perform the GS shader algorithm (along with any other required functions) and buffer results (either in the GRF or scratch memory) before issuing the FF_SYNC message. The issuing GS thread is stalled on the FF_SYNC writeback until it is that thread's turn to output VUEs. As only one GS thread at a time can output VUEs, the post-FF_SYNC output portion of the kernel should be optimized as much as possible to maximize parallelism.

Thread Execution

This topic is currently under development.

GS URB Entry

All outputs of a GS thread are stored in the single GS thread output URB entry. Cut (1 bit/vertex) or StreamID (2 bits/vertex) bits are packed into an optional 1-8 32B header. The **Control Data Format** and **Control Data Header Size** states specify the size and contents of the header data (if any).



Following the optional header is a variable number of 16B or 32B-aligned/granular vertices:

- When rendering is DISABLED, typically output vertices are 32B-aligned, with the exception of 16B-alignment for vertices <= 16B in length.
    - The absolute worst case size comes from three DW scalars output per vertex. If these are, say, three ".x" outputs, you need to store each DW in a 128b (16B) element, plus another pad 16B to keep the 32B alignment. So you require 4*16B = 64B/vertex. You have to have room for 1024 scalars / 3 scalar/vtx = 341 vertices. 341*64B = 21,824B. Then add 96B to hold 2b/vtx streamID and you get 21,920B entries.
- When rendering is ENABLED, each output vertex is 32B-aligned. Here the vertex header and vertex 'position' are required and therefore the minimum size vertex is 32B.
    - Here the worst case size isn't as bad as render-disabled, as you have to have a 4DW position output, plus any additional output. So, say you output 5 DW per vertex. You need 64B/vertex (16B vtx header, 16B position, 16B for the 2nd element, and 16B of pad). You have to have room for 1024 scalars / 5 = 204 vertices. 204*64 = 13,056B. Then add 64B to hold 2b/vtx streamID and you get 13,120B entries.

The size of the URB entry should be based on the declared maximum # of output vertices and the declared output vertex size (the union of per-stream vertex structures, if required).

GS URB Entry - Output Vertex Count

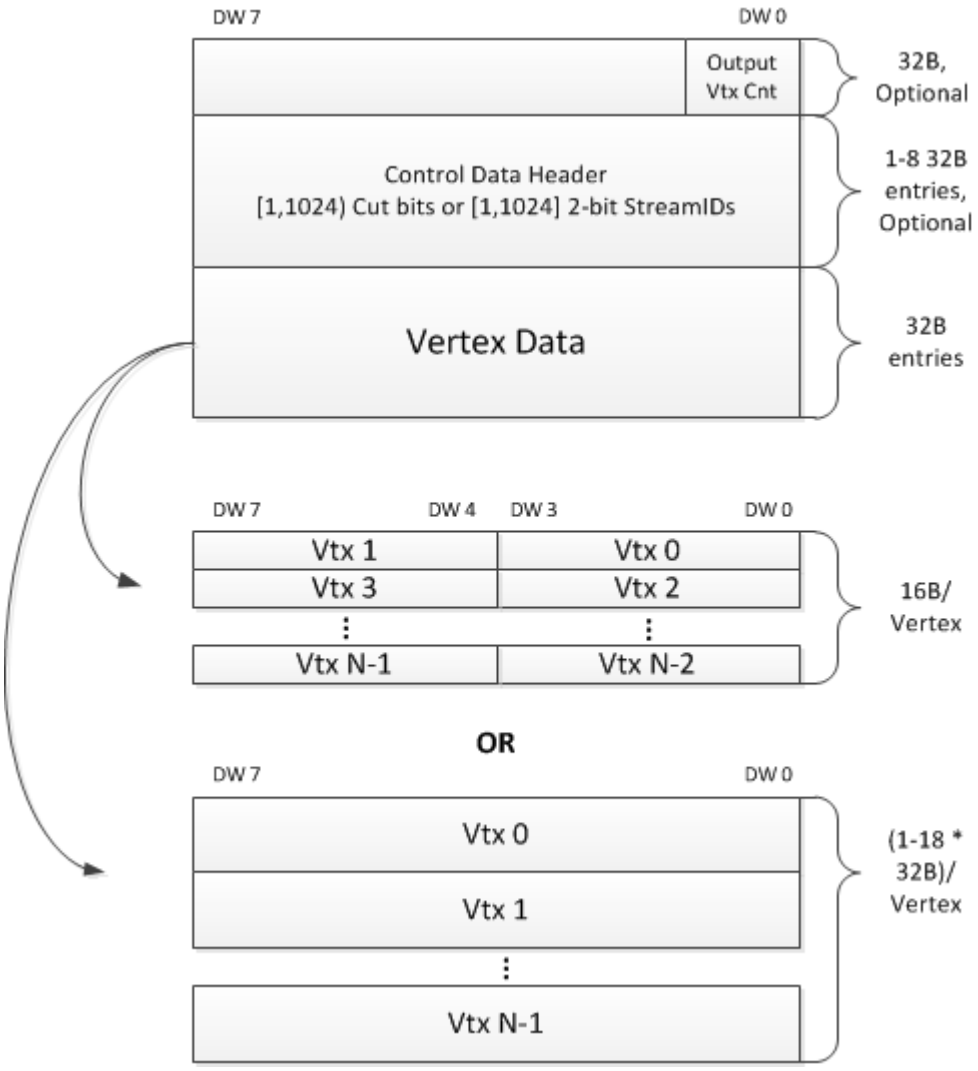
The GS URB entry is the same as in the two previous generations with the following exception: If **Static Output** (3DSTATE_GS) is clear, the URB entry starts with a 32B OUTPUT_VERTEX_COUNT structure as defined below. Ths control header (if present) immediately follows this structure. If **Static Output** is set, the control header (if present) appears at the very start of the URB entry (as described above).

GS OUTPUT_VERTEX_COUNT

DWord	Bit	Description
7:6	31:0	Reserved
0	31:16	Reserved
	15:0	**Output Vertex Count.** Indicates the number of vertices output from this GS shader invocation. Format = U16 Range: [0:1024]

This structure (if present) increases the maximum URB entry sizes (described above) by 32B.

The following diagram illustrates the possible layouts of a GS URB Entry:



GS Output Topologies

The following table lists which primitive topology types are valid for output by a GS thread.

PrimTopologyType	Supported for GS Thread Output?
LINELIST	Yes
LINELIST_ADJ	No
LINESTRIP	Yes
LINESTRIP_ADJ	No
LINESTRIP_BF	Yes
LINESTRIP_CONT	Yes
LINESTRIP_CONT_BF	Yes
LINELOOP	No
POINTLIST	Yes
POINTLIST_BF	Yes
POLYGON	Yes
QUADLIST	No
QUADSTRIP	No
RECTLIST	Yes
TRIFAN	Yes
TRIFAN_NOSTIPPLE	Yes
TRILIST	Yes
TRILIST_ADJ	No
TRISTRIP	Yes
TRISTRIP_ADJ	No
TRISTRIP_REV	Yes
PATCHLIST_xxx	Yes

GS Output StreamID

When the **GS Enable** is DISABLED, output vertices are assigned a StreamID = 0;

When the **GS Enable** is ENABLED, output vertices are assigned a StreamID = **Default StreamID** under the following conditions:

- **Control Data Format** = 0, or
- **Control Data Format** > 0 and **Control Data Format** = GSCTL_CUT

When the GS is enabled, **Control Data Format** > 0 and **Control Data Format** = GSCTL_SID, output vertices are assigned a StreamID as programmed in the Control Data output by the thread.

Primitive Output

(This section refers to output from the GS unit to the pipeline, not output from the GS thread)

The GS unit will output primitives (either passed-through or generated by a GS thread) in the proper order. This includes the buffering of a concurrent GS thread's output until the preceding GS thread terminates. Note that the requirement to buffer subsequent GS thread output until the preceding GS thread terminates has ramifications on determining the number of VUEs allocated to the GS unit and the number of concurrent GS threads allowed.

Statistics Gathering

There are a number of GS/StreamOutput pipeline statistics counters associated with the GS stage and GS threads. This subsection describes these counters and controls depending on device, even in the cases where functions outside of the GS stage (e.g., DataPort) are involved in the statistics gathering.

Refer to the *Statistics Gathering* summary provided earlier in this specification. Refer to the *Memory Interface Registers* chapter for details on these MMIO pipeline statistics counter registers, as well as the chapters corresponding to the other functions involved (e.g., DataPort, URB shared functions).

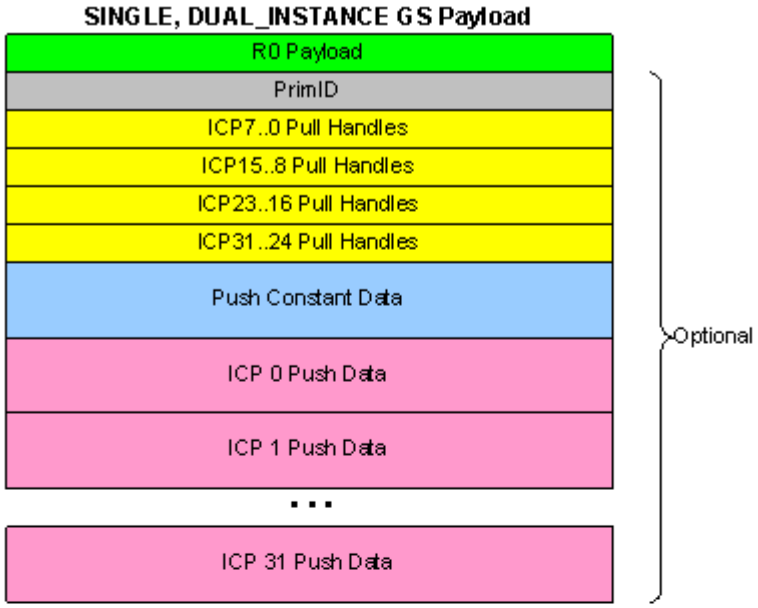
Payloads

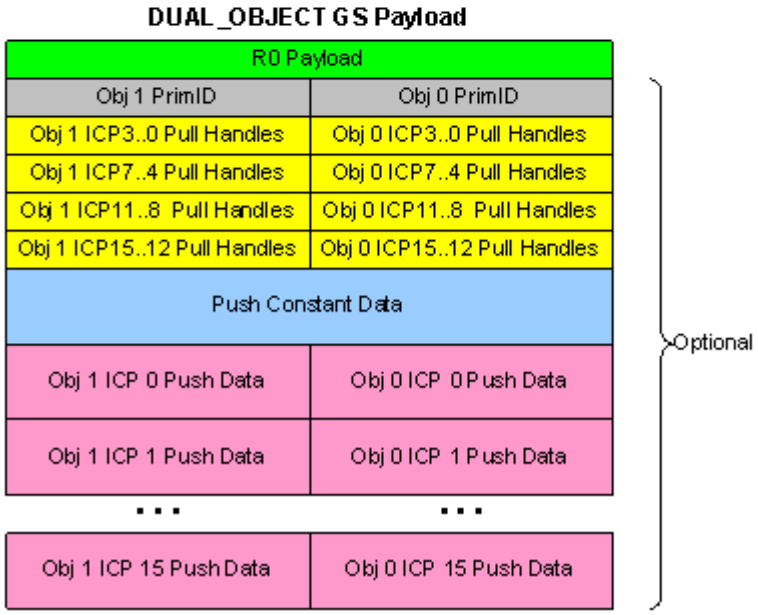
This topic is currently under development.

Thread Payload High-Level Layout

*Thread Payload High-Level Layout*shows the high-level layout of the payload delivered to GS threads.

GS Dispatch Layouts





Subsequent sections provide detailed layouts for different processor generations.

SIMD 4x2 Thread Payload

The table below shows the layout of the payload delivered to GS threads.

Refer to 3D Pipeline Stage Overview for details on fields that are common among the various pipeline stages.

GRF DWord	Bits	Description
R0.7	31	
	30:0	Reserved.
R0.6	31	**Dereference Thread.** This bit is defined to send back the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit, over some period of time. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer.** Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9.0	Reserved
	8:0	**FFTID.** This ID is assigned by the fixed function unit and is relative identifier for the thread. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved.
R0.3	31:5	**Sampler State Pointer.** Specifies the location of the Sampler State Table used by this thread, specified as a 32-byte granular offset from the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved.
	3:0	**Per Thread Scratch Space.**Specifies the amount of scratch space allowed for this thread. The value specifies the power that two is raised to (over determine the amount of scratch space). Programming Notes: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it. Format = U4 power of two (in excess of 10) Range = [0,11] indicating [1K Bytes, 2M Bytes]
R0.2	31:24	**Semaphore Index.** This is a DWord index used in URB_ATOMIC commands if the thread is using data pulled from input handles. This information is only required for pull-model vertex inputs

GRF DWord	Bits	Description
		and InstanceCount > 1. Format = U8
	23	Reserved.
	22	**Hint.** This is a copy of the corresponding 3DSTATE_GS bit. Format: U1
	21:16	**Primitive Topology Type.** This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV. If the **Discard Adjacency** bit is set, the topology type passed in the payload is UNDEFINED. Format: See *3D Pipeline*.
	15:13	Reserved
	12:0	**Semaphore Handle.** This is the URB offset pointing to the first GS semaphore DWord in the URB. Software is responsible for statically allocating the semaphore DWords in the URB. Refer to Semaphore Handle field in 3DSTATE_GS for size of semaphore allocation.
R0.1	31:27	**GS Instance ID 1.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. If "dispatch mode" is DUAL_OBJECT this field is not valid. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 1.** This is the URB offset where the EU's upper channels (DWords 7:4) results are stored. If only one object/instance is processed (shaded) by the thread, this field is effectively ignored (no results are stored for these channels, as controlled by the thread's Channel Mask).
R0.0	31:27	**GS Instance ID 0.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. If "dispatch mode" is DUAL_OBJECT, this field is not valid. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 0.** This is the URB offset where the EU's lower channels (DWords 3:0) results are stored.

GRF DWord	Bits	Description
The following register is included only if Include PrimitiveID is enabled.		
R1.7-R1.5	31:0	Reserved: MBZ.
R1.4	31:0	**Primitive ID 1**. This field contains the Primitive ID associated with (all instances) of input object 1. Only valid in DUAL_OBJECT mode. Format: U32
R1.3-R1.1	31:0	Reserved: MBZ.
R1.0	31:0	**Primitive ID 0.** This field contains the Primitive ID associated with (all instances) of input object 0. Format: U32
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled.		
Rn.7	31:16	ICP 7 Handle ID
	15:0	ICP 7 Handle
Rn.6	31:16	ICP 6 Handle ID
	15:0	ICP 6 Handle
Rn.5	31:16	ICP 5 Handle ID
	15:0	ICP 5 Handle
Rn.4	31:16	ICP 4 Handle ID
	15:0	ICP 4 Handle
Rn.3	31:16	ICP 3 Handle ID
	15:0	ICP 3 Handle
Rn.2	31:16	ICP 2 Handle ID
	15:0	ICP 2 Handle
Rn.1	31:16	ICP 1 Handle ID
	15:0	ICP 1 Handle
Rn.0	31:16	ICP 0 Handle ID
	15:0	ICP 0 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+1.7	31:16	ICP 15 Handle ID
	15:0	ICP 15 Handle
Rn+1.6	31:16	ICP 14 Handle ID
	15:0	ICP 14 Handle
Rn+1.5	31:16	ICP 13 Handle ID
	15:0	ICP 13 Handle
Rn+1.4	31:16	ICP 12 Handle ID

GRF DWord	Bits	Description
	15:0	ICP 12 Handle
Rn+1.3	31:16	ICP 11 Handle ID
	15:0	ICP 11 Handle
Rn+1.2	31:16	ICP 10 Handle ID
	15:0	ICP 10 Handle
Rn+1.1	31:16	ICP 9 Handle ID
	15:0	ICP 9 Handle
Rn+1.0	31:16	ICP 8 Handle ID
	15:0	ICP 8 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 15.		
Rn+2.7	31:16	ICP 23 Handle ID
	15:0	ICP 23 Handle
Rn+2.6	31:16	ICP 22 Handle ID
	15:0	ICP 22 Handle
Rn+2.5	31:16	ICP 21 Handle ID
	15:0	ICP 21 Handle
Rn+2.4	31:16	ICP 20 Handle ID
	15:0	ICP 20 Handle
Rn+2.3	31:16	ICP 19 Handle ID
	15:0	ICP 19 Handle
Rn+2.2	31:16	ICP 18 Handle ID
	15:0	ICP 18 Handle
Rn+2.1	31:16	ICP 17 Handle ID
	15:0	ICP 17 Handle
Rn+2.0	31:16	ICP 16 Handle ID
	15:0	ICP 16 Handle
The following register is included only if SINGLE or DUAL_INSTANCE mode and Include Vertex Handles is enabled and ICP Count > 23.		
Rn+3.7	31:16	ICP 31 Handle ID
	15:0	ICP 31 Handle
Rn+3.6	31:16	ICP 30 Handle ID
	15:0	ICP 30 Handle
Rn+3.5	31:16	ICP 29 Handle ID
	15:0	ICP 29 Handle
Rn+3.4	31:16	ICP 28 Handle ID

GRF DWord	Bits	Description
	15:0	ICP 28 Handle
Rn+3.3	31:16	ICP 27 Handle ID
	15:0	ICP 27 Handle
Rn+3.2	31:16	ICP 26 Handle ID
	15:0	ICP 26 Handle
Rn+3.1	31:16	ICP 25 Handle ID
	15:0	ICP 25 Handle
Rn+3.0	31:16	ICP 24 Handle ID
	15:0	ICP 24 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled.		
Rn.7	31:16	Object 1 ICP 3 Handle ID
	15:0	Object 1 ICP 3 Handle
Rn.6	31:16	Object 1 ICP 2 Handle ID
	15:0	Object 1 ICP 2 Handle
Rn.5	31:16	Object 1 ICP 1 Handle ID
	15:0	Object 1 ICP 1 Handle
Rn.4	31:16	Object 1 ICP 0 Handle ID
	15:0	Object 1 ICP 0 Handle
Rn.3	31:16	Object 0 ICP 3 Handle ID
	15:0	Object 0 ICP 3 Handle
Rn.2	31:16	Object 0 ICP 2 Handle ID
	15:0	Object 0 ICP 2 Handle
Rn.1	31:16	Object 0 ICP 1 Handle ID
	15:0	Object 0 ICP 1 Handle
Rn.0	31:16	Object 0 ICP 0 Handle ID
	15:0	Object 0 ICP 0 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 3.		
Rn+1.7	31:16	Object 1 ICP 7 Handle ID
	15:0	Object 1 ICP 7 Handle
Rn+1.6	31:16	Object 1 ICP 6 Handle ID
	15:0	Object 1 ICP 6 Handle
Rn+1.5	31:16	Object 1 ICP 5 Handle ID
	15:0	Object 1 ICP 5 Handle
Rn+1.4	31:16	Object 1 ICP 4 Handle ID
	15:0	Object 1 ICP 4 Handle

GRF DWord	Bits	Description
Rn+1.3	31:16	Object 0 ICP 7 Handle ID
	15:0	Object 0 ICP 7 Handle
Rn+1.2	31:16	Object 0 ICP 6 Handle ID
	15:0	Object 0 ICP 6 Handle
Rn+1.1	31:16	Object 0 ICP 5 Handle ID
	15:0	Object 0 ICP 5 Handle
Rn+1.0	31:16	Object 0 ICP 4 Handle ID
	15:0	Object 0 ICP 4 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 7.		
Rn+2.7	31:16	Object 1 ICP 11 Handle ID
	15:0	Object 1 ICP 11 Handle
Rn+2.6	31:16	Object 1 ICP 10 Handle ID
	15:0	Object 1 ICP 10 Handle
Rn+2.5	31:16	Object 1 ICP 9 Handle ID
	15:0	Object 1 ICP 9 Handle
Rn+2.4	31:16	Object 1 ICP 8 Handle ID
	15:0	Object 1 ICP 8 Handle
Rn+2.3	31:16	Object 0 ICP 11 Handle ID
	15:0	Object 0 ICP 11 Handle
Rn+2.2	31:16	Object 0 ICP 10 Handle ID
	15:0	Object 0 ICP 10 Handle
Rn+2.1	31:16	Object 0 ICP 9 Handle ID
	15:0	Object 0 ICP 9 Handle
Rn+2.0	31:16	Object 0 ICP 8 Handle ID
	15:0	Object 0 ICP 8 Handle
The following register is included only if DUAL_OBJECT mode and Include Vertex Handles is enabled and ICP Count > 11.		
Rn+3.7	31:16	Object 1 ICP 15 Handle ID
	15:0	Object 1 ICP 15 Handle
Rn+3.6	31:16	Object 1 ICP 14 Handle ID
	15:0	Object 1 ICP 14 Handle
Rn+3.5	31:16	Object 1 ICP 13 Handle ID
	15:0	Object 1 ICP 13 Handle
Rn+3.4	31:16	Object 1 ICP 12 Handle ID
	15:0	Object 1 ICP 12 Handle

GRF DWord	Bits	Description
Rn+3.3	31:16	Object 0 ICP 15 Handle ID
	15:0	Object 0 ICP 15 Handle
Rn+3.2	31:16	Object 0 ICP 14 Handle ID
	15:0	Object 0 ICP 14 Handle
Rn+3.1	31:16	Object 0 ICP 13 Handle ID
	15:0	Object 0 ICP 13 Handle
Rn+3.0	31:16	Object 0 ICP 12 Handle ID
	15:0	Object 0 ICP 12 Handle
Varies (optional)	31:0	Constant Data (optional): Some amount of constant data (possibly none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_GS command (taking the buffer enables into account). The Constant Data arrives in a non-interleaved format.
Varies	31:0	**Pushed Vertex Data.** There can be up to 32 vertices supplied, each with a size defined by the **Vertex URB Entry Read Length** state. The amount of data provided for each vertex is defined by the **Vertex URB Entry Read Length** state. For SINGLE or DUAL_INSTANCE dispatch modes, the pushed data for Vertex 0 immediately follows any pushed constant data. The pushed data for Vertex 1 immediately follows Vertex 0, and so on. There is no upper/lower swizzling of data. For DUAL_OBJECT dispatch mode, the pushed vertex data is split into upper and lower halves with Object 0 input vertices in the lower half, and Object 1 input vertices in the upper half.

SIMD8 Thread Payload

The table below shows the layout of the payload delivered to GS threads.

Refer to the **3D Pipeline Stage Overview section** for details on those fields that are common among the various pipeline stages.

GRF DWord	Bits	Description
R0.7	31	
	30:0	Reserved.
R0.6	31	**Dereference Thread.** This bit is defined to send the Handle ID back to HS to dereference the input handles for this thread.
	30:24	Reserved.
	23:0	**Thread ID.** This field uniquely identifies this thread within the threads spawned by this FF unit,

<table>
<tr><th>GRF DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td></td><td>over some period of time.
Format: Reserved for HW Implementation Use.</td></tr>
<tr><td rowspan="3">R0.5</td><td>31:10</td><td>Scratch Space Pointer. Specifies the location of the scratch space allocated to this thread, specified as a 1KB-aligned offset from the General State Base Address.
Format = GeneralStateOffset[31:10]</td></tr>
<tr><td>9.0</td><td>Reserved</td></tr>
<tr><td>8:0</td><td>FFTID. This ID is assigned by the fixed function unit and is a relative identifier for the thread. It is used to free up resources used by the thread upon thread completion.
Format:
<table><tr><th>Project</th><th>Format</th></tr><tr><td>CHV, BSW</td><td>U9</td></tr></table>Range:
<table><tr><th>Project</th><th>Range</th></tr><tr><td>CHV, BSW</td><td>0-503</td></tr></table></td></tr>
<tr><td rowspan="2">R0.4</td><td>31:5</td><td>Binding Table Pointer. Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the Surface State Base Address.
Format = SurfaceStateOffset[31:5]</td></tr>
<tr><td>4:0</td><td>Reserved.</td></tr>
<tr><td rowspan="3">R0.3</td><td>31:5</td><td>Sampler State Pointer. Specifies the location of the Sampler State Table used by this thread, specified as a 32-byte granular offset from the Dynamic State Base Address.
Format = DynamicStateOffset[31:5]</td></tr>
<tr><td>4</td><td>Reserved.</td></tr>
<tr><td>3:0</td><td>Per Thread Scratch Space. Specifies the amount of scratch space allowed to be used by this thread. The value specifies the power that two will be raised to (over determine the amount of scratch space).
Programming Notes: This amount is available to the kernel for information only. It is passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port ignores it.
Format = U4 power of two (in excess of 10)
Range = [0,11] indicating [1K Bytes, 2M Bytes]</td></tr>
<tr><td rowspan="2">R0.2</td><td>31:24</td><td>Reserved.</td></tr>
<tr><td>23</td><td>Reserved.</td></tr>
</table>

GRF DWord	Bits	Description
	22	**Hint.** This is a copy of the corresponding 3DSTATE_GS bit. Format: U1
	21:16	**Primitive Topology Type.** This field identifies the Primitive Topology Type associated with the primitive containing this object. It indirectly specifies the number of input vertices included in the thread payload. Note that the GS unit may toggle this value between TRISTRIP and TRISTRIP_REV. If the **Discard Adjacency**bit is set, the topology type passed in the payload is UNDEFINED. Format: See *3D Pipeline*.
	15:13	Reserved.
	12:0	Reserved.
R0.1-R0.0	31:0	Reserved.
R1.7	31:0	GS Instance ID / URB Return Handle for Object 7 (See R1.0)
R1.6	31:0	GS Instance ID / URB Return Handle for Object 6 (See R1.0)
R1.5	31:0	GS Instance ID / URB Return Handle for Object 5 (See R1.0)
R1.4	31:0	GS Instance ID / URB Return Handle for Object 4 (See R1.0)
R1.3	31:0	GS Instance ID / URB Return Handle for Object 3 (See R1.0)
R1.2	31:0	GS Instance ID / URB Return Handle for Object 2 (See R1.0)
R1.1	31:0	GS Instance ID / URB Return Handle for Object 1 (See R1.0)
R1.0	31:27	**GS Instance ID 0.** For each input object, the GS unit can spawn multiple threads (instances). This field starts at zero for the first instance of an object and increments for subsequent instances. Format: U5
	26:16	Reserved.
	15:0	**URB Return Handle 0.** This field is the URB offset where the EU lower channels (DWords 3:0) results are stored. Format: U14 64B-aligned URB Offset
The following register is included only if **Include PrimitiveID** is enabled.		
R2.7	31:0	**Primitive ID 7.** This field contains the Primitive ID associated with input object 7 (or the single input object if InstanceCount > 1). Format: U32
R2.6	31:0	**Primitive ID 6.** This field contains the Primitive ID associated with input object 6 (or the single input object if InstanceCount > 1). Format: U32

GRF DWord	Bits	Description
R2.5	31:0	**Primitive ID 5.** This field contains the Primitive ID associated with input object 5 (or the single input object if InstanceCount > 1). Format: U32
R2.4	31:0	**Primitive ID 4.** This field contains the Primitive ID associated with input object 4 (or the single input object if InstanceCount > 1). Format: U32
R2.3	31:0	**Primitive ID 3.** This field contains the Primitive ID associated with input object 3 (or the single input object if InstanceCount > 1). Format: U32
R2.2	31:0	**Primitive ID 2.** This field contains the Primitive ID associated with input object 2 (or the single input object if InstanceCount > 1). Format: U32
R2.1	31:0	**Primitive ID 1.** This field contains the Primitive ID associated with input object 1 (or the single input object if InstanceCount > 1). Format: U32
R2.0	31:0	**Primitive ID 0.** This field contains the Primitive ID associated with input object 0 (or the single input object if InstanceCount > 1). Format: U32
The following registers are included only if **Include Vertex Handles**is enabled and **InstanceCount** == 1.		
Rn.7	31:16	**Project** / **Description**: CHV, BSW / Object 7 ICP 0 Handle ID.
	15:0	**Object 7 ICP 0 Handle.**
Rn.6	31:16	**Project** / **Description**: CHV, BSW / Object 6 ICP 0 Handle ID.
	15:0	**Object 6 ICP 0 Handle.**
Rn.5	31:16	**Project** / **Description**: CHV, BSW / Object 5 ICP 0 Handle ID.
	15:0	**Object 5 ICP 0 Handle.**
Rn.4	31:16	**Project** / **Description**: CHV, BSW / Object 4 ICP 0 Handle ID.
	15:0	**Object 4 ICP 0 Handle.**

GRF DWord	Bits	Description
Rn.3	31:16	Project: CHV, BSW — Description: Object 3 ICP 0 Handle ID.
	15:0	**Object 3 ICP 0 Handle.**
Rn.2	31:16	Project: CHV, BSW — Description: Object 2 ICP 0 Handle ID.
	15:0	**Object 2 ICP 0 Handle.**
Rn.1	31:16	Project: CHV, BSW — Description: Object 1 ICP 0 Handle ID.
	15:0	**Object 1 ICP 1 Handle.**
Rn.0	31:16	Project: CHV, BSW — Description: Object 0 ICP 0 Handle ID.
	15:0	**Object 0 ICP 0 Handle.**
[Rn+1]	255:0	Project: CHV, BSW — Description: ICP 1 Handle/HandleID for Objects 0-7.
[Rn+2]	255:0	Project: CHV, BSW — Description: ICP 2 Handle/HandleID for Objects 0-7.
[Rn+3]	255:0	Project: CHV, BSW — Description: ICP 3 Handle/HandleID for Objects 0-7.
[Rn+4]	255:0	Project: CHV, BSW — Description: ICP 4 Handle/HandleID for Objects 0-7.
[Rn+5]	255:0	Project: CHV, BSW — Description: ICP 5 Handle/HandleID for Objects 0-7.
The following registers are included only if **Include Vertex Handles**is enabled and **InstanceCount** > 1.		
Rn.7	31:16	Reserved.
	15:0	Reserved.
Rn.6	31:16	Reserved.
	15:0	Reserved.
Rn.5	31:16	Project: CHV, BSW — Description: **ICP 5 Handle ID** (if required).
	15:0	**ICP 5 Handle** (if required).
Rn.4	31:16	Project: CHV, BSW — Description: **ICP 4 Handle ID** (if required).
	15:0	**ICP 4 Handle** (if required).

<table>
<tr><th>GRF DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td rowspan="2">Rn.3</td><td>31:16</td><td><table><tr><th>Project</th><th>Description</th></tr><tr><td>CHV, BSW</td><td>ICP 3 Handle ID (if required).</td></tr></table></td></tr>
<tr><td>15:0</td><td>ICP 3 Handle (if required).</td></tr>
<tr><td rowspan="2">Rn.2</td><td>31:16</td><td><table><tr><th>Project</th><th>Description</th></tr><tr><td>CHV, BSW</td><td>ICP 2 Handle ID (if required).</td></tr></table></td></tr>
<tr><td>15:0</td><td>ICP 2 Handle (if required).</td></tr>
<tr><td rowspan="2">Rn.1</td><td>31:16</td><td><table><tr><th>Project</th><th>Description</th></tr><tr><td>CHV, BSW</td><td>ICP 1 Handle ID (if required).</td></tr></table></td></tr>
<tr><td>15:0</td><td>ICP 1 Handle (if required).</td></tr>
<tr><td rowspan="2">Rn.0</td><td>31:16</td><td><table><tr><th>Project</th><th>Description</th></tr><tr><td>CHV, BSW</td><td>ICP 0 Handle ID (if required).</td></tr></table></td></tr>
<tr><td>15:0</td><td>ICP 0 Handle.</td></tr>
<tr><td>Varies

optional</td><td>31:0</td><td>Constant Data (optional):

Some amount of constant data (possibly none) can be extracted from the push constant buffer (PCB) and passed to the thread following the R0 Header. The amount of data provided is defined by the sum of the read lengths in the last 3DSTATE_CONSTANT_GS command (taking the buffer enables into account).

The Constant Data arrives in a non-interleaved format.</td></tr>
<tr><td>Varies</td><td></td><td>Pushed Vertex Data (InstanceCount == 1 Case):

Input data for the 8 input objects is located here. Object 0 (starting with Vertex 0 of Object 0) data is passed in DW0 of these GRFs, and Object 7 data is passed in DW7. The first GRF contains Vertex 0 Element 0 Component 0 for all 8 objects, followed by components 1-3 in the three subsequent GRFs. This is followed by GRFs containing Vertex 0 Element 1 (if it exists), and so on, up to the number of Vertex 0 elements specified by Vertex URB Read Length. This is followed by the data for Vertex 1 for all objects (if it exists), and so on until all relevant vertices are passed.

Note that the amount of data passed is limited by the number of GRFs supported by EUs. Software is responsible for comprehending this limit and resorting to the pull model as required.</td></tr>
<tr><td>Rv.7</td><td>31:0</td><td>Object 7 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.6</td><td>31:0</td><td>Object 6 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.5</td><td>31:0</td><td>Object 5 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.4</td><td>31:0</td><td>Object 4 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.3</td><td>31:0</td><td>Object 3 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.2</td><td>31:0</td><td>Object 2 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.1</td><td>31:0</td><td>Object 1 Vertex 0 Element 0 Component 0</td></tr>
<tr><td>Rv.0</td><td>31:0</td><td>Object 0 Vertex 0 Element 0 Component 0</td></tr>
</table>

GRF DWord	Bits	Description
Rv+1	31:0	**Object 0-7 Vertex 0 Element 0 Component 1**
		and so on
Varies		**Pushed Vertex Data** (InstanceCount > 1 Case): Input data for the single input object (shared across all instances) is located here. The pushed data for Vertex 0 immediately follows any pushed constant data. The pushed data for Vertex 1 immediately follows Vertex 0, and so on. There is no upper/lower swizzling of data.

Thread Request Generation

Once a FF unit determines that a thread can be requested, it must gather all the information required to submit the thread request to the Thread Dispatcher. This information is divided into several categories, as listed below and subsequently described in detail.

- **Thread Control Information:** This is the information required (from the FF unit) to establish the execution environment of the thread.
- **Thread Payload Header:** This is the first portion of the thread payload passed in the GRF, starting at GRF R0. This is information passed directly from the FF unit. It precedes the Thread Payload Input URB Data.
- **Thread Payload Input URB Data:** This is the second portion of the thread payload. It is read from the URB using entry handles supplied by the FF unit.

Thread Control Information

The following table describes the various state variables that a FF unit uses to provide information to the Thread Dispatcher and which affect the thread execution environment. Note that this information is not directly passed to the thread in the thread payload (though some fields may be subsequently accessed by the thread via architectural registers).

State Variables Included in Thread Control Information

State Variable	Usage	FFs
Kernel Start Pointer	This field, together with the **General State Pointer**, specifies the starting location (1st GEN4 core instruction) of the kernel program run by threads spawned by this FF unit. It is specified as a 64-byte-granular offset from the **General State Pointer**.	All FFs spawning threads
GRF Register Block Count	Specifies, in 16-register blocks, how many GRF registers are required to run the kernel. The Thread Dispatcher will only seek candidate EUs that have a sufficient number of GRF register blocks available. Upon selecting a target EU, the Thread DIspatcher will generate a logical-to-physical GRF mapping and provide this to the target EU.	All FFs spawning threads
Single Program Flow (SPF)	Specifies whether the kernel program has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). See CR0 description in *ISA Execution Environment*.	All FFs spawning threads
Thread Dispatch Priority	The Thread Dispatcher will give priority to those thread requests with Thread Dispatch Priority of HIGH_PRIORITY over those marked as LOW_PRIORITY. Within these two classes of thread requests, the Thread Dispatcher applies a priority order (e.g., round-robin --- though this algorithm is considered a device implementation-dependent detail).	All FFs spawning threads
Floating Point Mode	This determines the initial value of the **Floating Point Mode** bit of the EU's CR0 architectural register that controls floating point behavior in the EU core. (See ISA.)	All FFs spawning threads
Exceptions Enable	This bitmask controls the exception handing logic in the EU. (See ISA.)	All FFs spawning threads
Sampler Count	This is a hint which specifies how many indirect SAMPLER_STATE structures should be prefetched concurrent with thread initiation. It is recommended that software program this field to equal the number of samplers, though there may be some minor performance impact if this number gets large. This value should not exceed the number of samplers accessed by the thread as there would be no performance advantage. Note that the data prefetch is treated as any other memory fetch (with respect to page faults, etc.).	All stages supporting sampling (VS, GS, WM)
Binding Table Entry Count	This is a hint which specifies how many indirect BINDING_TABLE_STATE structures should be prefetched concurrent with thread initiation. (The notes included in Sampler Count (above) also apply to this field).	All FFs spawning threads

Thread Payload Generation

FF units are responsible for generating a thread *payload* – the data pre-loaded into the target EU's GRF registers (starting at R0) that serves as the primary direct input to a thread's kernel. The general format of these payloads follow a similar structure, though the exact payload size/content/layout is unique to each stage. This subsection describes the common aspects – refer to the specific stage's chapters for details on any differences.

The payload data is divided into two main sections: the *payload header* followed by the *payload URB data*. The payload header contains information passed directly from the FF unit, while the payload URB data is obtained from URB locations specified by the FF unit.

The first 256 bits of the thread payload (the initial contents of R0, aka "the R0 header") is specially formatted to closely match (and in some cases exactly match) the first 256 bits of thread-generated *messages* (i.e., the message header) accepted by shared functions. In fact, the send instruction supports having a copy of a GR's contents (such as R0) used as the message header. Software must take this intention into account (i.e., "don't muck with R0 unless you know what you're doing"). This is especially important given the fact that several fields in the R0 header are considered opaque to SW, where use or modification of their contents might lead to UNDEFINED results.

The payload header is further (loosely) divided into a leading *fixed payload header* section and a trailing, variable-sized *extended payload header* section. In general the size, content and layout of both payload header sections are FF-specific, though many of the fixed payload header fields are common amongst the FF stages. The extended header is used by the FF unit to pass additional information specific to that FF unit. The extended header is defined to start after the fixed payload header and end at the offset defined by **Dispatch GRF Start Register for URB Data**. Software can cause use the **Dispatch GRF Start Register for URB Data** field to insert padding into the extended header in order to maintain a fixed offset for the start of the URB data.

Fixed Payload Header

The payload header is used to pass *FF pipeline information* required as thread input data. This information is a mixture of SW-provided state information (state table pointers, etc.), primitive information received by the FF unit from the FF pipeline, and parameters generated/computed by the FF unit. Most of the fields of the fixed header are common between the FF stages. These non-FF-specific fields are described in Fixed Payload Header Fields (non-FF-specific). Note that a particular stage's header may not contain all these fields, so they are not "common" in the strictest sense.

Fixed Payload Header Fields (non-FF-specific)

Fixed Payload Header Field (non-FF-specific)	Description	FFs
FF Unit ID	Function ID of the FF unit. This value identifies the FF unit within the GEN4 subsystem. The FF unit uses this field (when transmitted in a Message Header to the URB Function) to detect messages emanating from its spawned threads.	All FFs spawning threads
Snapshot Flag		All FFs spawning threads
Thread ID	This field uniquely identifies this thread within the FF unit over some period.	All FFs spawning threads
Scratch Space Pointer	This is the starting location of the thread's allocated scratch space, specified as an offset from the **General State Base Address**. Note that scratch space is allocated by the FF unit on a per-thread basis, based on the **Scratch Space Base Pointer** and **Per-Thread Scratch Space Size** state variables. FF units assign a thread an arbitrarily-positioned region within this space. The scratch space for multiple (API-visible) entities (vertices, pixels) is interleaved within the thread's scratch space.	All FFs spawning threads
Dispatch ID	This field identifies this thread within the outstanding threads spawned by the FF unit. This field does *not* uniquely identify the thread over any significant period. **Implementation Note:** This field is effectively an "active thread index". It is used on a thread's URB allocation request to identify which thread's handle pool is to source the allocation. It is used upon thread termination to free up the thread's scratch space allocation.	All FFs spawning threads
Binding Table Pointer	This field, together with the **Surface State Base Pointer**, specifies the starting location of the Binding Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **Surface State Base Pointer**. See *Shared Functions* for a description of a Binding Table.	All FFs spawning threads
Sampler State Pointer	This field, together with the **General State Base Pointer**, specifies the starting location of the Sampler State Table used by threads spawned by the FF unit. It is specified as a 64-byte-granular offset from the **General State Base Pointer**. See *Shared Functions* for a description of a Sampler State Table.	**All FFs spawning threads which sample (VS, GS, WM)**
Per Thread Scratch Space	This field specifies the amount of scratch space allocated to each thread spawned by the FF unit. The driver must allocate enough contiguous scratch space, starting at the **Scratch Space Base Pointer**, to ensure that the **Maximum Number of**	All FFs spawning threads

Fixed Payload Header Field (non-FF-specific)	Description	FFs
	Threads can each get **Per-Thread Scratch Space** size without exceeding the driver-allocated scratch space.	
Handle ID <*n*>	This ID is assigned by the FF unit and links the thread to a specific entry within the FF unit. The FF unit will use this information upon detecting a URB_WRITE message issued by the thread. Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS, GS, CLIP, SF
URB Return Handle <*n*>	This is an initial destination URB handle passed to the thread. If the thread does output URB entries, this identifies the destination URB entry. Threads spawned by the GS, CLIP, and SF units are provided with a single Handle ID / URB Return Handle pair. Threads spawned by the VS are provided with one or two pairs (depending on how many vertices are to be processed). Threads spawned by the WM do not write to URB entries, and therefore this info is not supplied.	VS, GS, CLIP, SF
Primitive Topology Type	As part of processing an incoming primitive, a FF unit is often required to spawn a number of threads (for example, for each individual triangle in a TRIANGLE_STRIP). This field identifies the type of primitive which is being processed by the FF unit, and which has lead to the spawning of the thread. GEN4 kernels written to process different types of objects can use this value to direct that processing. E.g., when a CLIP kernel is to provide clipping for all the various primitive types, the kernel would need to examine the Primitive Topology Type to distinguish between point, lines, and triangle clipping requests. **Note:** In general, this field is identical to the Primitive Topology Type associated with the primitive vertices as received by the FF unit. Refer to the individual FF unit chapters for cases where the FF unit modifies the value before passing it to the thread. (for example, certain units perform toggling of TRIANGLESTRIP and TRIANGLESTRIP_REV).	GS, CLIP, SF, WM

Extended Payload Header

The extended header is of variable-size, where inclusion of a field is determined by FF unit state programming.

In order to permit the use of common kernels (thus reducing the number of kernels required), the **Dispatch GRF Start Register for URB Data** state variable is supported in all FF stages. This SV is used to place the payload URB data at a specific starting GRF register, irrespective of the size of the extended header. A kernel can therefore reference the payload URB data at fixed GRF locations, while conditionally referencing extended payload header information.

Payload URB Data

In each thread payload, following the payload header, is some amount of URB-sourced data required as input to the thread. This data is divided into an optional *Constant URB Entry* (CURBE), following either by a Primitive URB Entry (WM) or a number of Vertex URB Entries (VS, GS, CLIP, SF). A FF unit only knows the location of this data in the URB, and is never exposed to the contents. For each URB entry, the FF unit will supply a sequence of handles, read offsets and read lengths to the GEN4 subsystem. The subsystem will read the appropriate 256-bit locations of the URB, optionally perform swizzling (VS only), and write the results into sequential GRF registers (starting at **Dispatch GRF Start Register for URB Data**).

State Variables Controlling Payload URB Data

State Variable	Usage	FFs
Dispatch GRF Start Register for URB Data	This SV identifies the starting GRF register receiving payload URB data. Software is responsible for ensuring that URB data does not overwrite the Fixed or Extended Header portions of the payload.	FFs spawning threads
Vertex URB Entry Read Offset	This SV specifies the starting offset within VUEs from which vertex data is to be read and supplied in this stage's payloads. It is specified as a 256-bit offset into any and all VUEs passed in the payload. This SV can be used to skip over leading data in VUEs that is not required by the stage's threads (e.g., skipping over the Vertex Header data at the SF stage, as that information is not required for setup calculations). Skipping over irrelevant data can only help to improve performance. Specifying a vertex data source extending beyond the end of a vertex entry is UNDEFINED.	VS, GS
Vertex URB Entry Read Length	This SV determines the amount of vertex data (starting at **Vertex URB Entry Read Offset**) to be read from each VUEs and passed into the payload URB data. It is specified in 256-bit units. A zero value is INVALID (at very least one 256-bit unit must be read). Specifying a vertex data source extending beyond the end of a VUE is UNDEFINED.	

Programming Restrictions: (others may already been mentioned)

- The maximum size payload for any thread is limited by the number of GRF registers available to the thread, as determined by min(128, 16 * **GRF Register Block Count**). Software is responsible for ensuring this maximum size is not exceeded, taking into account:
  - The size of the Fixed and Extended Payload Header associated with the FF unit.
  - The **Dispatch GRF Start Register for URB** Data SV.
  - The amount of CURBE data included (via **Constant URB Entry Read Length**)
  - The number of VUEs included (as a function of FF unit, it's state programming, and incoming primitive types)
  - The amount of VUE data included for each vertex (via **Vertex URB Entry Read Length**)
  - (For WM-spawned PS threads) The amount of Primitive URB Entry data.
- For any type of URB Entry reads:
  - Specifying a source region (via Read Offset, Read Length) that goes past the end of the URB Entry allocation is illegal.
    - The allocated size of Vertex/Primitive URB Entries is determined by the **URB Entry Allocation Size** value provided in the pipeline state descriptor of the FF unit owning the VUE/PUE.
    - The allocated size of CURBE entries is determined by the **URB Entry Allocation Size** value provided in the CS_URB_STATE command.

Stream Output Logic (SOL) Stage

The Stream Output Logic (SOL) stage receives 3D topologies originating in the VF, DS or GS stage. If enabled, the SOL stage uses programmed state information to copy portions of the vertex data associated with the incoming topologies across one or more Stream Output (SO) Buffers.

State

This section contains state commands and structures pertaining to the StreamOut Logic (SOL) stage of the 3D pipeline.

3DSTATE_STREAMOUT

The 3DSTATE_STREAMOUT command specifies control information for the SOL stage. Included are enables and sizes for input streams and enables for output buffers.

The SOL unit incorrectly double buffers MMIO/NP registers and only moves them into the design for usage when control topology is received with the SOL unit dirty state.

If the state does not change, need to resend the same state.

There is no need to send a pipeline state update to the SOL unit after SOL unit MMIO registers or non-pipeline state are written.

3DSTATE_STREAMOUT

3DSTATE_SO_DECL_LIST Command

The 3DSTATE_SO_DECL_LIST instruction defines a list of Stream Output (SO) declaration entries (SO_DECLs) and associated information for all specific SO streams in parallel.

3DSTATE_SO_DECL_LIST

SO_DECL

3DSTATE_SO_BUFFER

The 3DSTATE_SO_BUFFER command specifies the location and characteristics of an SO buffer in memory.

3DSTATE_SO_BUFFER

The SOL Unit also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_WM, 3DSTATE_PS_EXTRA, and 3DSTATE_DEPTH_STENCIL_STATE fields.

Signal [CHV, BSW]	Description	Formula
SOL_INT::Render_Enable	If clear, the SO stage will not forward any topologies down the pipeline. If set, the SO stage will forward topologies associated with Render Stream Select down the pipeline. This bit is used even if SO Function	= (3DSTATE_STREAMOUT::Force_Rending == Force_ ((3DSTATE_STREAMOUT::Force_Rending != Force_O !(3DSTATE_GS::Enable && 3DSTATE_GS::Output Ve Size == 0) && !3DSTATE_STREAMOUT::API_Render_Disable && (3DSTATE_DEPTH_STENCIL_STATE::Stencil_TestEn ||

	Enable is DISABLED.	3DSTATE_DEPTH_STENCIL_STATE::Depth_TestEnab 3DSTATE_DEPTH_STENCIL_STATE::Depth_WriteEna 3DSTATE_PS_EXTRA::PS_Valid || 3DSTATE_WM::Legacy Depth_Buffer_Clear || 3DSTATE_WM::Legacy Depth_Buffer_Resolve_Ena 3DSTATE_WM::Legacy Hierarchical_Depth_Buffer_Resolve_Enable))

DW1[21]	DW1[20]	Stream Offset	Action
Full legacy mode. HW doesn't LOAD or STORE, it simply updates the MMIO register during stream out. SW can can the LOAD/STORE using MI_LOAD_REG/ MI_STORE_REG.			
0	0	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = no action
0	0	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = no action
SW can cause the LOAD of the SO_OFFSET using MI_LOAD_REG, and HW performs the STORE.			
0	1	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = No action, write SO_WRITE_OFFSET[x] to memory
0	1	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = No action, write SO_WRITE_OFFSET[x] to memory
HW performs the LOAD, and SW can cause the STOREs using MI_STORE_REG_MEM.			
1	0	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = stream offset
1	0	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = load from memory
HW performs both the LOAD and STORE.			
1	1	not equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = stream offset, write SO_WRITE_OFFSET[x] to memory
1	1	equal to 0xFFFFFFFF	SO_WRITE_OFFSET[x] = load from memory, write SO_WRITE_OFFSET[x] to memory

"SO_WRITE_OFFSET[x] =" occurs before the execution of the primitive, while write SO_WRITE_OFFSET[x] to memory occurs after the execution of the primitive.

Functions

This topic is currently under development.

Input Buffering

For the purposes of stream output, the SOL stage breaks incoming topologies into independent objects without adjacency information. In the process, any adjacent-only vertices are ignored. For example, it converts TRISTRIP_ADJ into independent 3-vertex triangles. However, if rendering is enabled, incoming topologies are passed to the Clip stage unmodified and therefore the Clip unit must be enabled if there is any possibility of "ADJ" topologies reaching it.

Note that the SOL unit will not see incomplete objects: the VF will remove incomplete input objects, the GS will remove GS-generated incomplete objects, and the DS does not output incomplete objects as only complete topologies are generated by the TE stage.

The OSB (Object Staging Buffer) reorders the vertices of odd-numbered triangles in TRISTRIP topologies to match API requirements.

Incoming topologies are tagged with a 2-bit StreamID. The StreamID is 0 for topologies originating from the VF stage (i.e., 3DPRIMITIVE_xxx). For topologies output from the GS stage, the StreamID is set by the GS shader. A Stream *n* Vertex Length is associated with each stream, and defines how much data is read from the URB for vertices in that stream.

The following table specifies how the SOL stage streams out object vertices for each incoming topology type.

PrimTopologyType	Order of Vertices Streamed Out	
<PRIMITIVE_TOPOLOGY> (N = # of vertices)	[<object#>] = (<vert#>,...);	Any SOL Notes
POINTLIST POINTLIST_BF	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
LINELIST (N is multiple of 2)	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
LINELIST_ADJ (N is multiple of 4)	[0] = (1,2); [1] = (5,6); ...; [(N/4)-1)] = (N-3,N-2)	
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF (N >= 2)	[0] = (0,1); [1] = (1,2); ...; [N-2] = (N-2,N-1)	

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT (N >= 4)	[0] = (1,2); [1] = (2,3); ...; [N-4] = (N-3,N-2)	LINESTRIP_ADJ_CONT is added. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINELOOP	N/A	Not supported after VF.
TRILIST (N is multiple of 3)	[0] = (0,1,2); [1] = (3,4,5); ...; [(N/3)-1] = (N-3,N-2,N-1)	
RECTLIST, RECTLIST_SUBPIXEL	Same as TRILIST	Handled same as TRILIST.
TRILIST_ADJ (N is multiple of 6)	[0] = (0,2,4); [1] = (6,8,10); ...; [(N/6)-1] = (N-6,N-4,N-2)	
TRISTRIP (N >= 3) REORDER_LEADING	[0] = (0,1,2); [1] = (1,3,2); [k even] = (k,k+1,k+2) [k odd] = (k,k+2,k+1) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP (N >= 3) REORDER_TRAILING	[0] = (0,1,2); [1] = (2,1,3); [k even] = (k,k+1,k+2) [k odd] = (k+1,k,k+2) [N-3] = (see above)	"Odd" triangles have vertices reordered to yield increasing trailing vertices starting with v2.
TRISTRIP_REV (N >= 3) REORDER_LEADING	[0] = (0,2,1) [1] = (1,2,3);...; [k even] = (k,k+2,k+1) [k odd] = (k,k+1,k+2) [N-3] = (see above)	"Even" triangles have vertices reordered to yield increasing leading vertices starting with v0.
TRISTRIP_REV (N >= 3) REORDER_TRAILING	[0] = (1,0,2) [1] = (1,2,3);...; [k even] = (k+1,k,k+2) [k odd] = (k,k+1,k+2)	"Even" triangles have vertices reordered to yield increasing trailing vertices starting with v2.

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
	[N-3] = (see above)	
TRISTRIP_ADJ (N even, N >= 6) REORDER_LEADING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (2,6,4); ...; N > 10: [0] = (0,2,4); [1] = (2,6,4); ...; [k>1, even] = (2k, 2k+2, 2k+4); [k>2, odd] = (2k, 2k+4, 2k+2);...; Trailing object: [(N/2)-3, even] = (N-6,N-4,N-2); [(N/2)-3, odd] = (N-6,N-2,N-4);	"Odd" objects have vertices reordered to yield increasing-by-2 leading vertices starting with v0.
TRISTRIP_ADJ (N even, N >= 6) REORDER_TRAILING	N = 6 or 7: [0] = (0,2,4) N = 8 or 9: [0] = (0,2,4); [1] = (4,2,6); ...; N > 10: [0] = (0,2,4); [1] = (4,2,6); ...; [k>1, even] = (2k, 2k+2, 2k+4); [k>2, odd] = (2k+2,2k, 2k+4,);...; Trailing object: [(N/2)-3, even] = (N-6,N-4,N-2); [(N/2)-3, odd] = (N-4,N-6,N-2);	"Odd" objects have vertices reordered to yield increasing-by-2 trailing vertices starting with v4.
TRIFAN (N > 2)	[0] = (0,1,2); [1] = (0,2,3); ...; [N-3] = (0, N-2, N-1);	
TRIFAN_NOSTIPPLE	Same as TRIFAN	
POLYGON, POLYGON_CONT	Same as TRIFAN	POLYGON_CONT is added. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-

PrimTopologyType <PRIMITIVE_TOPOLOGY> (N = # of vertices)	Order of Vertices Streamed Out [<object#>] = (<vert#>,...);	Any SOL Notes
		empted 3DPRIMITIVE.
QUADLIST QUADSTRIP	N/A	Not supported after VF.
PATCHLIST_1	[0] = (0); [1] = (1); ...; [N-2] = (N-2);	
PATCHLIST_2	[0] = (0,1); [1] = (2,3); ...; [(N/2)-1] = (N-2,N-1)	
PATCHLIST_3..32	similar to above	

Stream Output Function

As previously mentioned, incoming 3D topologies are targeted at one of the four streams. The SOL stage contains state information specific to each of the four streams.

A stream's list of SO declarations (SO_DECL structures) is used to perform the SO function for objects targeted to that particular stream. The 3DSTATE_SO_DECL_LIST command is used to specify the list of SO_DECL structures for all four streams in parallel. Software is required to scan the SO_DECL lists of streams to determine which SO buffers are targeted. The Stream To Buffer Selects bits in 3DSTATE_SO_DECL_LIST must be programmed accordingly (if the buffer is targeted, the select bit must set, else it must be cleared).

If a stream has no SO_DECL state defined (NumEntries is 0), incoming objects targeting that stream are effectively ignored. As there is no attempt to perform stream output, overflow detection is neither required nor performed.

Otherwise, an overflow check is performed. First any attempt to output to a disabled buffer is detected. This occurs when the stream has a Stream To Buffer Selects bit set but the corresponding SO Buffer Enable is clear. Assuming all targeted buffers are enabled, an additional check is made to ensure that there is enough room in each targeted buffer to hold the number of vertices which be output to it (for the input object). Here the buffer's current end address is compared to what the write offset would be if the output was performed. The latter value is computed as (write_offset + vertex_count * buffer_pitch). If this value is greater than the end address, an overflow is signalled. This check is performed for each buffer included in Stream To Buffer Selects.

If an overflow is not signaled, the SO function is performed. The SO_DECL list for the targeted stream is traversed independently for each object vertex, and the operation specified by the SO_DECL structure is performed (typically causing data to be appended to an SO buffer). In the process, SO buffer Write Offsets are incremented.

Stream Output Buffers

Up to four SO buffers are supported. The SO buffer parameters (start/end address, etc.) are specified by the 3DSTATE_SO_BUFFER command.

The 3DSTATE_STREAMOUT command specifies an SO Buffer Enable bit for each of the buffers. If a buffer is disabled, its state is ignored and no output will be attempted for that buffer. Any attempt to output to that buffer will immediately signal an overflow condition.

The SOL stage maintains a current Write Offset register value for each SO buffer. These registers can be written via MI_LOAD_REGISTER_MEM or MI_LOAD_REGISTER_IMM commands. The SOL stage will increment the Write Offsets as a part of the SO function. Software can cause a Write Offset register to be written to memory via an MI_STORE_REGISTER_MEM command, though a preceding flush operation may be required to ensure that any previous SO functions have completed.

Surface Format Name
R32G32B32A32_FLOAT
R32G32B32A32_SINT
R32G32B32A32_UINT
R32G32B32_FLOAT
R32G32B32_SINT
R32G32B32_UINT
R32G32_FLOAT
R32G32_SINT
R32G32_UINT
R32_SINT
R32_UINT
R32_FLOAT

Rendering Disable

Independent of SOL function enable, if rendering (i.e, 3D pipeline functions past the SOL stage) is enabled (via clearing the Rendering Disable bit), the SOL stage will pass topologies for a specific input stream (as selected by Render Stream Select) down the pipeline, with the exception of PATCHLIST_n topologies which are never passed downstream. Software must ensure that the vertices exiting the SOL stage include a vertex header and position value so that the topologies can be correctly processed by subsequent pipeline stages. Specifically, rendering must be disabled whenever 128-bit vertices are output from a GS thread.

If Rendering Disable is set, the SOL stage will prevent any topologies from exiting the SOL stage.

Statistics

The SOL stage controls the incrementing of two 64-bit statistics counter registers for each of the four output buffer slots, SO_NUM_PRIMS_WRITTEN[] and SO_PRIM_STORAGE_NEEDED[].

3D Pipeline Rasterization

This topic is currently under development.

Common Rasterization State

This section contains rasterization state pointers.

3DSTATE_VIEWPORT_STATE_POINTERS_CC

3DSTATE_VIEWPORT_STATE_POINTERS_SF_CLIP

3DSTATE_SCISSOR_STATE_POINTERS

3DSTATE_RASTER

3D Pipeline – CLIP Stage Overview

The CLIP stage of the GEN 3D Pipeline is similar to the GS stage in that it can be used to perform general processing on incoming 3D objects via spawned GEN4 threads. However, the CLIP stage also includes specialized logic to perform a *ClipTest* function on incoming objects. These two usage models of the CLIP stage are outlined below.

Refer to the *Common 3D FF Unit Functions* subsection in the *3D Overview* chapter for a general description of a 3D Pipeline stage, as much of the CLIP stage operation and control falls under these "common" functions. I.e., many of the CLIP stage state variables and CLIP thread payload parameters are described in *3D Overview*, and although they are listed here for completeness, that chapter provides the detailed description of the associated functions.

Refer to this chapter for an overall description of the CLIP stage, details on the ClipTest function, and any exceptions the CLIP stage exhibits with respect to common FF unit functions.

Clip Stage – 3D Clipping

The ClipTest fixed function is provided to optimize the CLIP stage for support of generalized *3D Clipping*. The CLIP FF unit examines the position of incoming vertices, performs a fixed function *VertexClipTest* on these positions, and then examines the results for the vertices of each independent object in *ClipDetermination*.

The results of ClipDetermination indicate whether an object is to be clipped (MustClip), discarded (TrivialReject) or passed down the pipeline unmodified (TrivialAccept). In the MustClip case, the fixed function clipping hardware is responsible for performing the actual 3D Clipping algorithm. The CLIP hardware is passed the source object vertex data and is able to output a new, arbitrary 3D primitive (e.g., the clipped primitive), or no output at all. Note that the output primitive is comprised of newly-generated vertex positions, barycentric attributes and shares vertices with the source primitive for rest of the attributes.The CLIP unit maintains the proper ordering of CLIP-generated primitives and any surrounding trivially-accepted primitives and processes all the primitives in order.

The outgoing primitive stream is sent down the pipeline to the Strip/Fan (SF) FF stage (now including the read-back VUE Vertex Header data such as Vertex Position (NDC or screen space), RTAIndex, VPIndex, PointWidth) and control information (PrimType, PrimStart, PrimEnd) while the remainder of the vertex data remains in the VUE in the URB.

Fixed Function Clipper

For CHV, BSW, the GPU supports Fixed Function Clipping.

Note: In an earlier generation, clipping was done in the EU. However the clipper thread latency was high and caused a bottleneck in the pipeline. Hence the motivation for a fixed function clipper.

Concepts

This section provides an overview of 3D clip-testing and clipping concepts, as defined by the Direct3D* and OpenGL* APIs. It is provided as background material. Some of the concepts impact HW functionality while others impact CLIP kernel functionality.

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

CLIP Stage Input

As a stage of the GEN 3D pipeline, the CLIP stage receives inputs from the previous (GS) stage. Refer to *3D Overview* for an overview of the various types of input to a 3D Pipeline stage. The remainder of this subsection describes the inputs specific to the CLIP stage.

State

This section contains state clips for the Clip Stage. For each processor generation, the state used by the clip stage is defined by the appropriate inline state packet, linked below.

3DSTATE_CLIP

3D_STATE_CLIP

The Clip unit will unconditionally reject incoming PATCHLIST topologies, if not already discarded by SOL. So there is no need for SW to explicitly set the CLIP_mode to reject PATCHLIST topologies.

Clip Unit also receives 3DSTATE_RASTER. It also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_VS, 3DSTATE_DS and 3DSTATE_GS fields.

Signal [CHV, BSW]	Description	Formula
CLIP_INT::Front_Winding	Determines whether a triangle object is considered “front facing” if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.	= 3DSTATE_RASTER::FrontWinding
CLIP_INT::CullMode	Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points, or rectangles.	= 3DSTATE_RASTER::CullMode
CLIP_INT::Viewport Z ClipTest Enable	This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.	= 3DSTATE_RASTER::Viewport Z ClipTest Enable
CLIP_INT::User Clip Distance Cull Test Enable Bitmask	This 8-bit mask field selects which of the 8 user clip distances against which trivial reject/trivial accept determination needs to be made (does not cause a must clip).	`= (3DSTATE_CLIP::ForceUser Clip Distance Cull Test Enable Bitmask == Force) ? 3DSTATE_CLIP::User Clip Distance Cull Test Enable Bitmask :`

	DX10 allows simultaneous use of ClipDistance and Cull Distance test of up to 8 distances. This mask must be mutually exclusive to final CLIP_INT::User Clip Distance Clip Test Enable Bitmask. Same mask bit can't be set for both.	3DSTATE_GS::GS_Enable ? 3DSTATE_GS:: GS User Clip Distance Cull Test Enable Bitmask : 3DSTATE_DS:DS_Enable ? 3DSTATE_DS:: DS User Clip Distance Cull Test Enable Bitmask : 3DSTATE_INT:VS_Enable ? 3DSTATE_VS:: VS User Clip Distance Cull Test Enable Bitmask : 0
CLIP_INT::User Clip Distance Clip Test Enable Bitmask	This 8-bit mask field selects which of the 8 user clip distances against which trivial reject/trivial accept determination needs to be made (does not cause a must clip). DX10 allows simultaneous use of ClipDistance and Clip Distance test of up to 8 distances. This mask must be mutually exclusive to final CLIP_INT::User Clip Distance Cull Test Enable Bitmask. Same mask bit can't be set for both.	= (3DSTATE_CLIP::ForceUser Clip Distance Clip Test Enable Bitmask == Force) ? 3DSTATE_CLIP::User Clip Distance Clip Test Enable Bitmask : 3DSTATE_GS:GS_Enable ? 3DSTATE_GS::GS User Clip Distance Clip Test Enable Bitmask : 3DSTATE_DS::DS_Enable ? 3DSTATE_DS::DS User Clip Distance Clip Test Enable Bitmask : 3DSTATE_VS::VS_Enable ? 3DSTATE_VS::VS User Clip Distance Clip Test Enable Bitmask : 0

VUE Readback

Starting with the CLIP stage, the 3D pipeline requires vertex information in addition to the VUE handle. For example, the CLIP unit's VertexClipTest function needs the vertex position, as does the SF unit's functions. This information is obtained by the 3D pipeline reading a portion of each vertex's VUE data directly from the URB. This readback (effectively) occurs immediately before the CLIP VertexClipTest function, and immediately after a CLIP thread completes the output of a destination VUE.

The Vertex Header (first 256 bits) of the VUE data is read back. (See the previous *VUE Formats* subsection (above) for details on the content and format of the Vertex Header.) Additional Clip/Cull data (located immediately past the Vertex Header) may be read prior to clipping.

This readback occurs automatically and is not under software control. The only software implication is that the Vertex Header must be valid at the readback points, and therefore must have been previously loaded or written by a thread.

VertexClipTest Function

The VertexClipTest function compares each incoming vertex position (x,y,z,w) with various viewport and guardband parameters (either hard-coded values or specified by state variables).

The RHW component of the incoming vertex position is tested for NaN value, and if a NaN is detected, the vertex is marked as "BAD" by setting the outcode[BAD]. If a NaN is detected in any vertex homogeneous x,y,z,w component or an enabled ClipDistance value, the vertex is marked as "BAD" by setting the outcode[BAD].

In general, any object containing a BAD vertex will be discarded, as how to clip/render such objects is undefined.

However, in the case of CLIP_ALL mode, a CLIP thread will be spawned even for objects with "BAD" vertices. The CLIP kernel is required to handle this case, and can examine the **Object Outcode [BAD]** payload bit to detect the condition. (Note that the VP and GB Object Outcodes are UNDEFINED when BAD is set.)

If the incoming RHW coordinate is negative (including negative 0) the NEGW outcode is set. Also, this condition is used to select the proper comparison functions for the VP and GB outcode tests (below).

Next, the VPXY and GB outcodes are computed, depending on the corresponding enable SV bits. If one of VPXY or GB is disabled, the enabled set of outcodes are copied to the disabled set of outcodes. This effectively defines the disabled boundaries to coincide with the enabled boundaries (i.e., disabling the GB is just like setting it to the VPXY values, and vice versa).

The VPZ outcode is computed as required by the API mode SV.

Finally, the incoming UserClipFlags are masked and copied to corresponding outcodes.

The following algorithm is used by VertexClipTest:

```
//
// Vertex ClipTest
//
// On input:
// if (CLIP.PreMapped)
//   x,y are viewport mapped
//   z is NDC ([0,1] is visible)
// else
//   x,y,z are NDC (post-perspective divide)
//   w is always 1/w
//
// Initialize outCodes to "inside"
//
outCode[*] = 0
//
// Check if w is NaN
// Any object containing one of these "bad" vertices will likely be discarded
//
if (ISNAN(homogeneous x,y,z,w or enabled ClipDistance value)
{
   outCode[BAD] = 1
}
//
// If 1/w is negative, w is negative and therefore outside of the w=0 plane
//
```

```
//
rhw_neg = ISNEG(rhw)
if (rhw_neg)
{
   outCode[NEGW] = 1
}
//
// View Volume Clip Test
// If Premapped, the 2D viewport is defined in screen space
// otherwise the canonical NDC viewvolume applies ([-1,1])
//
if (CLIP_STATE.PreMapped)
{
   vp_XMIN = CLIP_STATE.VP_XMIN
   vp_XMAX = CLIP_STATE.VP_XMAX
   vp_YMIN = CLIP_STATE.VP_YMIN
   vp_YMAX = CLIP_STATE.VP_YMAX
} else {
   vp_XMIN = -1.0f
   vp_XMAX = +1.0f
   vp_YMIN = -1.0f
   vp_YMAX = +1.0f
}

if (CLIP_STATE.ViewportXYClipTestEnable) {
   outCode[VP_XMIN] = (x < vp_XMIN)
   outCode[VP_XMAX] = (x > vp_XMAX)
   outCode[VP_YMIN] = (y < vp_YMIN)
   outCode[VP_YMAX] = (y > vp_YMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
      outCode[VP_XMIN] = (x >= vp_XMIN)
      outCode[VP_XMAX] = (x <= vp_XMAX)
      outCode[VP_YMIN] = (y >= vp_XMIN)
      outCode[VP_YMAX] = (y <= vp_XMAX)
   }
   #endif
   if (rhw_neg) {
      outCode[VP_XMIN] = (x > vp_XMIN)
      outCode[VP_XMAX] = (x < vp_XMAX)
      outCode[VP_YMIN] = (y > vp_XMIN)
      outCode[VP_YMAX] = (y < vp_XMAX)
   }
}

if (CLIP_STATE.ViewportZClipTestEnable) {
   if (CLIP_STATE.APIMode == APIMODE_D3D) {
      vp_ZMIN = 0.0f
      vp_ZMAX = 1.0f
   } else { // OGL
      vp_ZMIN = -1.0f
      vp_ZMAX = 1.0f
   }
   outCode[VP_ZMIN] = (z < vp_ZMIN)
   outCode[VP_ZMAX] = (z > vp_ZMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
      outCode[VP_ZMIN] = (z >= vp_ZMIN)
      outCode[VP_ZMAX] = (z <= vp_ZMAX)
   }
   #endif
```

```
    if (rhw_neg) {
       outCode[VP_ZMIN] = (z > vp_ZMIN)
       outCode[VP_ZMAX] = (z < vp_ZMAX)
    }
}
//
// Guardband Clip Test
//
if {CLIP_STATE.GuardbandClipTestEnable) {
   gb_XMIN = CLIP_STATE.Viewport[vpindex].GB_XMIN
   gb_XMAX = CLIP_STATE.Viewport[vpindex].GB_XMAX
   gb_YMIN = CLIP_STATE.Viewport[vpindex].GB_YMIN
   gb_YMAX = CLIP_STATE.Viewport[vpindex].GB_YMAX
   outCode[GB_XMIN] = (x < gb_XMIN)
   outCode[GB_XMAX] = (x > gb_XMAX)
   outCode[GB_YMIN] = (y < gb_YMIN)
   outCode[GB_YMAX] = (y > gb_YMAX)

   #ifdef (BW-E0)
   if (rhw_neg) {
      outCode[GB_XMIN] = (x >= gb_XMIN)
      outCode[GB_XMAX] = (x <= gb_XMAX)
      outCode[GB_YMIN] = (y >= gb_YMIN)
      outCode[GB_YMAX] = (y <= gb_YMAX)
   }
   #endif
   if (rhw_neg) {
      outCode[GB_XMIN] = (x > gb_XMIN)
      outCode[GB_XMAX] = (x < gb_XMAX)
      outCode[GB_YMIN] = (y > gb_YMIN)
      outCode[GB_YMAX] = (y < gb_YMAX)
   }
}
//
// Handle case where either VP or GB disabled (but not both)
// In this case, the disabled set take on the outcodes of the enabled set
//
if (CLIP_STATE.ViewportXYClipTestEnable && !CLIP_STATE.GuardbandClipTestEnable) {
   outCode[GB_XMIN] = outCode[VP_XMIN]
   outCode[GB_XMAX] = outCode[VP_XMAX]
   outCode[GB_YMIN] = outCode[VP_YMIN]
   outCode[GB_YMAX] = outCode[VP_YMAX]
} else if (!CLIP_STATE.ViewportXYClipTestEnable && CLIP_STATE.GuardbandClipTestEnable) {
   outCode[VP_XMIN] = outCode[GB_XMIN]
   outCode[VP_XMAX] = outCode[GB_XMAX]
   outCode[VP_YMIN] = outCode[GB_YMIN]
   outCode[VP_YMAX] = outCode[GB_YMAX]
}
//
// X/Y/Z NaN Handling
//
xyorgben = (CLIP_STATE.ViewportXYClipTestEnable || CLIP_STATE.GuardbandClipTestEnable)
if (isNAN(x)) {
   outCode[GB_XMIN] = xyorgben
   outCode[GB_XMAX] = xyorgben
   outCode[VP_XMIN] = xyorgben
   outCode[VP_XMAX] = xyorgben
}
if (isNAN(y)) {
   outCode[GB_YMIN] = xyorgben
   outCode[GB_YMAX] = xyorgben
   outCode[VP_YMIN] = xyorgben
   outCode[VP_YMAX] = xyorgben
```

```
}
if (isNaN) {
   outCode[VP_ZMIN] = CLIP_STATE.ViewportZClipTestEnable
   outCode[VP_ZMAX] = CLIP_STATE.ViewportZClipTestEnable
}
//
// UserClipFlags
//
ExamineUCFs
for (i=0; i<7; i++)
{
   outCode[UC0+i] = userClipFlag[i] & CLIP_STATE.UserClipFlagsClipTestEnableBitmask[i]
}
outCode[UC7] = userClipFlag[i] & CLIP_STATE.UserClipFlagsClipTestEnableBitmask[7]
```

Object Staging

The CLIP unit's Object Staging Buffer (OSB) accepts streams of input vertex information packets, along with each vertex's VertexClipTest result (outCode). This information is buffered until a complete object or the last vertex of the primitive topology is received. The OSB then performs the ClipDetermination function on the object vertices, and takes the actions required by the results of that function.

Partial Object Removal

The OSB is responsible for removing incomplete LINESTRIP and TRISTRIP objects that it may receive from the preceding stage (GS). Partial object removal is not supported for other primitive types due to either (a) the GS is not permitted to output those primitive types (e.g., primitives with adjacency info), and the VF unit will have removed the partial objects as part of 3DPRIMITIVE processing, or (b) although the GS thread is allowed to output the primitive type (e.g., LINELIST), it is assumed that the GS kernel will be correctly implemented to avoid outputting partial objects (or pipeline behavior is UNDEFINED).

An object is considered 'partial' if the last vertex of the primitive topology is encountered (i.e., PrimEnd is set) before a complete set of vertices for that object have been received. Given that only LINESTRIP and TRISTRIP primitive types are subject to CLIP unit partial object removal, the only supported cases of partial objects are 1-vertex LINESTRIPs and 1 or 2-vertex TRISTRIPs.

ClipDetermination Function

In ClipDetermination, the vertex outcodes of the primitive are combined in order to determine the clip status of the object (TR: trivially reject; TA: trivial accept; MC: must clip; BAD: invalid coordinate). Only those vertices included in the object are examined (3 vertices for a triangle, 2 for a line, and 1 for a point). The outcode bit arrays for the vertices are separately ANDed (intersection) and ORed (union) together (across vertices, not within the array) to yield objANDCode and objORCode bit arrays.

TR/TA against interesting boundary subsets are then computed. The TR status is computed as the logical OR of the appropriate objANDCode bits, as the vertices need only be outside of one common boundary to be trivially rejected. The TA status is computed as the logical NOR of the appropriate objORCode bits, as any vertex being outside of any of the boundaries prevents the object from being trivially accepted.

If any vertex contains a BAD coordinate, the object is considered BAD and any computed TR/TA results will effectively be ignored in the final action determination.

Next, the boundary subset TR/TA results are combined to determine an overall status of the object. If the object is TR against any viewport or enabled UC plane, the object is considered TR. Note that, by definition, being TR against a VPXY boundary implies that the vertices will be TR agains the corresponding GB boundary, so computing TR_GB is unnecessary.

The treatment of the UCF outcodes is conditional on the UserClipFlags MustClip Enable state. If DISABLED, an object that is not TR against the UCFs is considered TA against them. Put another way, objects will only be culled (not clipped) with respect to the UCFs. If ENABLED, the UCF outcodes are treated like the other outcodes, in that they are used to determine TR, TA or MC status, and an object can be passed to a CLIP thread simply based on it straddling a UCF.

Finally, the object is considered MC if it is neither TR or TA.

The following logic is used to compute the final TR, TA, and MC status.

```
//
// ClipDetermination
//
// Compute objANDCode and objORCode
//
switch (object type) {
   case POINT:
      {
         objANDCode[...] = v0.outCode[...]
         objORCode[...]  = v0.outCode[...]
      } break
   case LINE:
      {
         objANDCode[...] = v0.outCode[...] & v1.outCode[...]
         objORCode[...]  = v0.outCode[...] | v1.outCode[...]
      } break
   case TRIANGLE:
      {
         objANDCode[...] = v0.outCode[...] & v1.outCode[...] & v2.outCode[...]
         objORCode[...]  = v0.outCode[...] | v1.outCode[...] | v2.outCode[...]
      } break
}
//
// Determine TR/TA against interesting boundary subsets
//
TR_VPXY = (objANDCode[VP_L] | objANDCode[VP_R] | objANDCode[VP_T] | objANDCode[VP_B])
TR_GB   = (objANDCode[GB_L] | objANDCode[GB_R] | objANDCode[GB_T] | objANDCode[GB_B])
TA_GB   = !(objORCode[GB_L] | objORCode[GB_R]  | objORCode[GB_T]  | objORCode[GB_B])
TA_VPZ  = !(objORCode[VP_N] | objORCode[VP_Z])
TR_VPZ  = (objANDCode[VP_N] | objANDCode[VP_Z])
TA_UC   = !(objORCode[UC0]  | objORCode[UC1]   | ... | objORCode[UC7])
TR_UC   = (objANDCode[UC0]  | objANDCode[UC1]  | ... | objANDCode[UC7])
BAD     = objORCode[BAD]
TA_NEGW = !objORCode[NEGW]
TR_NEGW = objANDCode[NEGW]

//
// Trivial Reject
//
// An object is considered TR if all vertices are TR against any common boundary
// Note that this allows the case of the VPXY being outside the GB
```

```
//
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC || TR_NEGW
#else
TR = TR_GB || TR_VPXY || TR_VPZ || TR_UC

//
// Trivial Accept
//
// For an object to be TA, it must be TA against the VPZ and GB, not TR,
// and considered TA against and NEGW
// If the UCMC mode is disabled, an object is considered TA against the UC
// as long as it isn't TR against the UC.
// If the UCMC mode is enabled, then the object really has to be TA against the UC
// to be considered TA
// In this way, enabling the UCMC mode will force clipping if the object is neither
// TA or TR against the UC
//
TA   = !TR && TA_GB && TA_VPZ && TA_NEGW
UCMC = CLIP_STATE.UserClipFlagsMustClipEnable
TA   = TA && ( (UCMC && TA_UC) || (!UCMC && !TR_UC) )

//
// MustClip
// This is simply defined as not TA or TR
// Note that exactly one of TA, TR and MC will be set
//
MC = !(TA || TR)
```

ClipMode State

The ClipMode state determines what action the CLIP unit takes given the results of ClipDetermination. The possible actions are:

- **PASSTHRU:** Pass the object directly down the pipeline. A CLIP thread is not spawned.
- **DISCARD:** Remove the object from the pipeline and dereference object vertices as required (that is, dereferencing will not occur if the vertices are shared with other objects).
- **SPAWN:** Pass the object to a CLIP thread. In the process of initiating the thread, the object vertices may be dereferenced.

The following logic is used to determine what to do with the object (PASSTHRU or DISCARD or SPAWN).

```
//
// Use the ClipMode to determine the action to take
//
switch (CLIP_STATE.ClipMode) {
   case NORMAL:
      {
         PASSTHRU = TA && !BAD
         DISCARD = TR || BAD
         SPAWN   = MC && !BAD
      }
   case CLIP_ALL:
      {
         PASSTHRU = 0
         DISCARD  = 0
         SPAWN    = 1
      }
   case CLIP_NOT_REJECT:
```

```
        {
           PASSTHRU = 0
           DISCARD  = TR   || BAD
           SPAWN    = !(TR || BAD)
        }
    case REJECT_ALL:
        {
           PASSTHRU = 0
           DISCARD  = 1
           SPAWN    = 0
        }
    case ACCEPT_ALL:
        {
           PASSTHRU = !BAD
           DISCARD  = BAD
           SPAWN    = 0
        }
 } endswitch
```

NORMAL ClipMode

In NORMAL mode, objects will be discarded if TR or BAD, passed through if TA, and passed to a CLIP thread if MC. Those mode is typically used when the CLIP kernel is only used to perform 3D Clipping (the expected usage model).

CLIP_ALL ClipMode

In CLIP_ALL mode, all objects (regardless of classification) will be passed to CLIP threads. Note that this includes BAD objects. This mode can be used to perform arbitrary processing in the CLIP thread, or as a backup if for some reason the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are not sufficient for controlling 3D Clipping.

CLIP_NON_REJECT ClipMode

This mode is similar to CLIP_ALL mode, but TR and BAD objects are discarded an all other (TA, MC) objects are passed to CLIP threads. Usage of this mode assumes that the CLIP unit fixed functions (VertexClipTest, ClipDetermination) are sufficient at least in respect to determining trivial reject.

REJECT_ALL ClipMode

In REJECT_ALL mode, all objects (regardless of classification) are discarded. This mode effectively clips out all objects.

ACCEPT_ALL ClipMode

In ACCEPT_ALL mode, all non-BAD objects are passed directly down the pipeline. This mode partially disables the CLIP stage. BAD objects will still be discarded, and incomplete primitives (generated by a GS thread) will be discarded.

Primitive topologies with adjacency are also handled, in that the adjacent-only vertices are dereferenced and only non-adjacent objects are passed down the pipeline. This condition can arise when primitive

topologies with adjacency are generated but the GS stage is disabled. If this condition is allowed, the CLIP stage must not be completely disabled – as this would allow adjacent vertices to pass through the CLIP stage and lead to unpredictable results as the rest of the pipeline does not comprehend adjacency.

Object Pass-Through

Depending on ClipMode, objects may be passed directly down the pipeline. The PrimTopologyType associated with the output objects may differ from the input PrimTopologyType, as shown in the table below.

Programming Note: The CLIP unit does *not* tolerate primitives with adjacency that have "dangling vertices". This should not be an issue under normal conditions, as the VF unit does not generate these sorts of primitives and the GS thread is restricted (though by specification only) to not output these sorts of primitives.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
POINTLIST	POINTLIST	
POINTLIST_BF	POINTLIST_BF	
LINELIST	LINELIST	
LINELIST_ADJ	LINELIST	Adjacent vertices removed.
LINESTRIP	LINESTRIP	
LINESTRIP_ADJ, LINESTRIP_ADJ_CONT	LINESTRIP	Adjacent vertices removed. LINESTRIP_ADJ_CONT is added. LINESTRIP_ADJ_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
LINESTRIP_BF	LINESTRIP_BF	
LINESTRIP_CONT	LINESTRIP_CONT	
LINESTRIP_CONT_BF	LINESTRIP_CONT_BF	
LINELOOP	N/A	Not supported after GS.
TRILIST	TRILIST	
RECTLIST	RECTLIST	
TRILIST_ADJ	TRILIST	Adjacent vertices removed.
TRISTRIP	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects See Tristrip Clipping subsection.
TRISTRIP_REV	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects. See Tristrip Clipping subsection.

Input PrimTopologyType	Pass-Through Output PrimTopologyType	Notes
TRISTRIP_ADJ	TRISTRIP or TRISTRIP_REV	Depends on where the incoming strip is broken (if at all) by discarded or clipped objects. Adjacent vertices removed. See Tristrip Clipping subsection.
TRIFAN	TRIFAN	
TRIFAN_NOSTIPPLE	TRIFAN_NOSTIPPLE	
POLYGON, POLYGON_CONT	POLYGON	POLYGON_CONT is added. POLYGON_CONT is generated by the Vertex Fetch unit on a restore of a mid-draw pre-empted 3DPRIMITIVE.
QUADLIST	N/A	Not supported after GS.
QUADSTRIP	N/A	Not supported after GS.

Primitive Output

(This section refers to output from the CLIP unit to the pipeline, not output from the CLIP thread)

The CLIP unit will output primitives (either passed-through or generated by a CLIP thread) in the proper order. This includes the buffering of a concurrent CLIP thread's output until the preceding CLIP thread terminates. Note that the requirement to buffer subsequent CLIP thread output until the preceding CLIP thread terminates has ramifications on determining the number of VUEs allocated to the CLIP unit and the number of concurrent CLIP threads allowed.

Other Functionality

Introduction for this section under development.

Statistics Gathering

The CLIP unit includes logic to assist in the gathering of certain pipeline statistics . The statistics take the form of MI counter registers (see Memory Interface Registers), where the CLIP unit provides signals causing those counters to increment.

Software is responsible for controlling (enabling) these counters in order to provide the required statistics at the DDI level. For example, software might need to disable statistics gathering before submitting non-API-visible objects (e.g., RECTLISTs) for processing.

The CLIP unit must be ENABLED (via the CLIP Enable bit of PIPELINED_STATE_POINTERS) for it to affect the statistics counters. This might lead to a pathological case where the CLIP unit needs to be ENABLED simply to provide statistics gathering. If no clipping functionality is desired, Clip Mode can be set to ACCEPT_ALL to effectively inhibit clipping while leaving the CLIP stage ENABLED.

The statistic the CLIP unit affects (if enabled) is CL_INVOCATION_COUNT, incremented for every object received from the GS stage.

CL_INVOCATION_COUNT

If the **Statistics Enable** bit (CLIP_STATE) is set, the CLIP unit increments the CL_INVOCATION_COUNT register for every complete object received from the GS stage.

To maintain a count of application-generated objects, software must clear the CLIP unit's **Statistic Enable** whenever driver-generated objects are rendered.

3D Pipeline - Strips and Fans (SF) Stage

The Strips and Fan (SF) stage of the 3D pipeline is responsible for performing "setup" operations required to rasterize 3D objects.

This functionality is handled completely in hardware, and the SF unit no longer has the ability to spawn threads.

Inputs from CLIP

The following table describes the per-vertex inputs passed to the SF unit from the previous (CLIP) stage of the pipeline.

SF's Vertex Pipeline Inputs

Variable	Type	Description
primType	enum	Type of primitive topology the vertex belongs to. *Primitive Assembly* for a list of primitive types supported by the SF unit. See *3D Pipeline* for descriptions of these topologies. Notes: The CLIP unit will convert any primitive with adjacency (3DPRIMxxx_ADJ) it receives from the pipeline into the corresponding primitive without adjacency (3DPRIMxxx). QUADLIST, QUADSTRIP, LINELOOP primitives are not supported by the SF unit. Software must use a GS thread to convert these to some other (supported) primitive type. [CHV, BSW] If an object is clipped by the hardware clipper, the CLunit would force this field to "3DPRIM_POLYGON". SFunit would process this incoming object just as it would any other "3DPRIM_POLYGON". SFunit selects vertex 0 as the provoking vertex.
primStart,primEnd	boolean	Indicate vertex's position within the primitive topology
vInX[]	float	Vertex X position (screen space or NDC space)
vInY[]	float	Vertex Y position (screen space or NDC space)
vInZ[]	float	Vertex Z position (screen space or NDC space)
vInInvW[]	float	Reciprocal of Vertex homogeneous (clip space) W
hVUE[]	URB address	Points to the vertex's data stored in the URB (one VUE handle per vertex)
renderTargetArrayIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in subsequent processing.

Variable	Type	Description
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object. If this vertex is the leading vertex of an object within the primitive topology, this value will be associated with that object in the Viewport Transform and Scissor subfunctions, otherwise the value is ignored. Note that for primitive topologies with vertices shared between objects, this means a shared vertex may be subject to multiple Viewport Transformation operations if the viewPortIndex varies within the topology.
pointSize	uint	If this vertex is within a POINTLIST[_BF] primitive topology, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

Attribute Setup/Interpolation Process

The following sections describe the Attribute Setup/Interpolation Process.

Attribute Setup/Interpolation Process

Hardware computes all needed parameters, as there is no setup thread.

Outputs to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects. The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Coefficients for Z, 1/W, perspective and non-perspective b1 and b2 per vertex, and attribute vertex deltas a0, a1, and a2 per attribute.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.).
- Edge or line interpolation information (e.g., edge equation coefficients, etc.).
- Information on where the WM is to start rasterization of the object.
- Object orientation (front/back-facing).
- Last Pixel indication (for line drawing).

Primitive Assembly

The first subfunction within the SF unit is *Primitive Assembly*. Here 3D primitive vertex information is buffered and, when a sufficient number of vertices are received, converted into basic 3D objects which are then passed to the Viewport Transformation subfunction.

The number of vertices passed with each primitive is constrained by the primitive type. *Primitive Assembly*. Passing any other number of vertices results in UNDEFINED behavior. Note that this restriction only applies to primitive output by GS threads (which is under control of the GS kernel). See the Vertex Fetch chapter for details on how the VF unit automatically removes incomplete objects resulting from processing a 3DPRIMITIVE command.

SF-Supported Primitive Types & Vertex Count Restrictions

primType	VertexCount Restriction
3DPRIM_TRILIST	nonzero multiple of 3
3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE	>=3
3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	>=3
3DPRIM_LINELIST	nonzero multiple of 2
3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	>=2
3DPRIM_RECTLIST	nonzero multiple of 3
3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	nonzero

Primitive Assembly for a list of the 3D object types.

3D Object Types

objectType	generated by primType	Vertices/Object
3DOBJ_POINT	3DPRIM_POINTLIST 3DPRIM_POINTLIST_BF	1
3DOBJ_LINE	3DPRIM_LINELIST 3DPRIM_LINESTRIP 3DPRIM_LINESTRIP_CONT 3DPRIM_LINESTRIP_BF 3DPRIM_LINESTRIP_CONT_BF	2
3DOBJ_TRIANGLE	3DPRIM_TRILIST 3DPRIM_TRISTRIP 3DPRIM_TRISTRIP_REVERSE 3DPRIM_TRIFAN 3DPRIM_TRIFAN_NOSTIPPLE 3DPRIM_POLYGON	3
3DOBJ_RECTANGLE	3DPRIM_RECTLIST	3 (expanded to 4 in RectangleCompletion)

Primitive Assembly for the outputs of Primitive Decomposition.

Primitive Decomposition Outputs

Variable	Type	Description
objectType	enum	Type of object. *Primitive Assembly*
nV	uint	The number of object vertices passed to Object Setup. *Primitive Assembly*
v[0..nV-1]*	various	Data arrays associated with object vertices. Data in the array consists of X, Y, Z, invW and a pointer to the other vertex attributes. These additional attributes are not used by directly by the 3D fixed functions but are made available to the SF thread. The number of valid vertices depends on the object type. *Primitive Assembly*
invertOrientation	enum	Indicates whether the orientation (CW or CCW winding order) of the vertices of a triangle object should be inverted. Ignored for non-triangle objects.
backFacing	enum	Valid only for points and line objects, indicates a back facing object. This is used later for culling.
provokingVtx	uint	Specifies the index (into the *v[]* arrays) of the vertex considered the "provoking" vertex (for flat shading). The selection of the provoking vertex is programmable via SF_STATE (**xxx Provoking Vertex Select** state variables.)
polyStippleEnable	boolean	TRUE if Polygon Stippling is enabled. FALSE for TRIFAN_NOSTIPPLE. Ignored for non-triangle objects.
continueStipple	boolean	Only applies to line objects. TRUE if Line Stippling should be continued (i.e., not reset) from where the previous line left off. If FALSE, Line Stippling is reset for each line object.
renderTargetIndex	uint	Index of the render target (array element or 3D slice), clamped to 0 by the GS unit if the max value was exceeded. This value is simply passed in SF thread payloads and not used within the SF unit.
viewPortIndex	uint	Index of a viewport transform matrix within the SF_VIEWPORT structure used to perform Viewport Transformation on object vertices and scissor operations on an object.
pointSize	unit	For point objects, this value specifies the screen space size (width,height) of the square point to be rasterized about the vertex position. Otherwise the value is ignored.

The following table defines, for each primitive topology type, which vertex's VPIndex/RTAIndex applies to the objects within the topology.

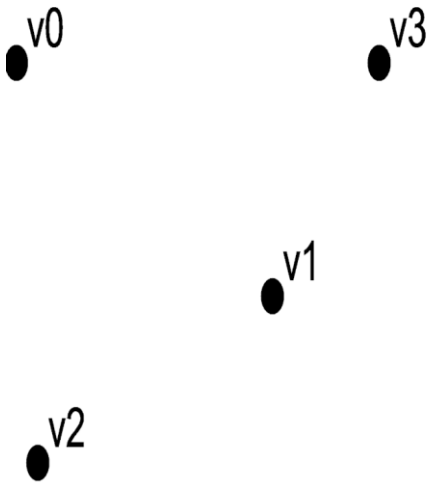
VPIndex/RTAIndex Selection

PrimTopologyType	Viewport Index Usage
POINTLIST POINTLIST_BF	Each vertex supplies the VPIndex for the corresponding point object
LINELIST	The leading vertex of each line supplies the VPIndex for the corresponding line object. V0.VPIndex→ Line(V0,V1) V2.VPIndex→ Line(V2,V3) ...
LINESTRIP LINESTRIP_BF LINESTRIP_CONT LINESTRIP_CONT_BF	The leading vertex of each line segment supplies the VPIndex for the corresponding line object. V0.VPIndex→ Line(V0,V1) V1.VPIndex→ Line(V1,V2) ... NOTE: If the VPIndex changes within the topology, shared vertices will be processed (mapped) multiple times.
TRILIST RECTLIST	The leading vertex of each triangle/rect supplies the VPIndex for the corresponding triangle/rect objects. V0.VPIndex→ Tri(V0,V1,V2) V3.VPIndex→ Tri(V3,V4,V5) ...
TRISTRIP TRISTRIP_REVERSE	The leading vertex of each triangle supplies the VPIndex for the corresponding triangle object. V0.VPIndex→ Tri(V0,V1,V2) V1.VPIndex→ Tri(V1,V2,V3) ... NOTE: If the VPIndex changes within the primitive, shared vertices will be processed (mapped) multiple times.
TRIFAN TRIFAN_NOSTIPPLE POLYGON	The first vertex (V0) supplies the VPIndex for all triangle objects.

Point List Decomposition

The 3DPRIM_POINTLIST and 3DPRIM_POINTLIST_BACKFACING primitives specify a list of independent points.

3DPRIM_POINTLIST Primitive



The decomposition process divides the list into a series of basic 3DOBJ_POINT objects that are then passed individually and in order to the Object Setup subfunction. The *provokingVertex* of each object is, by definition, v[0].

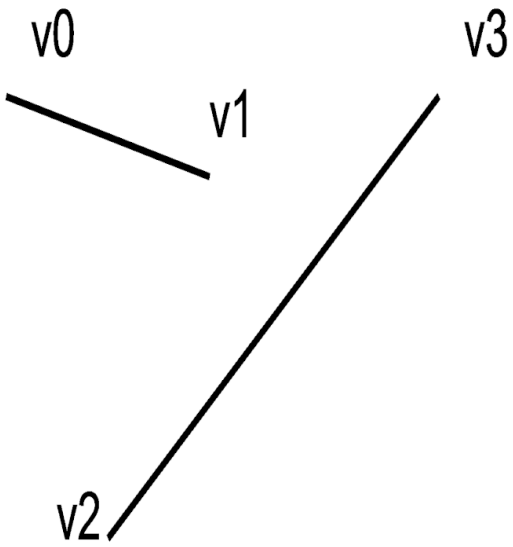
Points have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing points. Primitives of type 3DPRIM_POINTLIST_BACKFACING are decomposed exactly the same way as 3DPRIM_POINTLIST primitives, but the *backFacing* variable is set for resulting point objects being passed on to object setup.

```
PointListDecomposition()
{
   objectType = 3DOBJ_POINT
   nV = 1
   provokingVtx = 0
   if (primType == 3DPRIM_POINTLIST)
      {
         backFacing = FALSE
      }
   else // primType == 3DPRIM_POINTLIST_BACKFACING
      {
         backFacing = TRUE
      }

   for each (vertex in [0..vertexCount-1])
      {
         v[0] ← vIn[i] // copy all arrays
                       // (for example, v[]X, v[]Y, and so on)
         ObjectSetup()
      }
}
```

Line List Decomposition

The 3DPRIM_LINELIST primitive specifies a list of independent lines.

3DPRIM_LINELIST Primitive



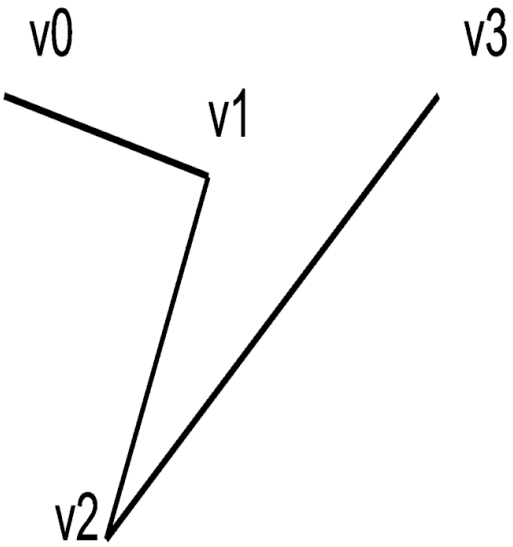
The decomposition process divides the list into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0, v1; v2, v3; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```
LineListDecomposition()
   {
      objectType = 3DOBJ_LINE
      nV = 2
      provokingVtx = Line List/Strip Provoking Vertex Select       continueStipple = FALSE
      for each (vertex I in [0..vertexCount-2] by 2)
         {
             v[0] arrays ← vIn[i] arrays
             v[1] arrays ← vIn[i+1] arrays
             ObjectSetup()
         }
   }
```

Line Strip Decomposition

The 3DPRIM_LINESTRIP, 3DPRIM_LINESTRIP_CONT, 3DPRIM_LINESTRIP_BF, and 3DPRIM_LINESTRIP_CONT_BF primitives specify a list of connected lines.

3DPRIM_LINESTRIP_xxx Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_LINE objects that are then passed individually and in order to the Object Setup stage. The lines are generated with the following object vertex order: v0,v1; v1,v2; and so on. The *provokingVertex* of each object is taken from the **Line List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

Lines have no winding order, so the primitive command is used to explicitly state whether they are back-facing or front-facing lines. Primitives of type 3DPRIM_LINESTRIP[_CONT]_BF are decomposed exactly the same way as 3DPRIM_LINESTRIP[_CONT] primitives, but the *backFacing* variable is set for the resulting line objects being passed on to object setup. Likewise 3DPRIM_LINESTRIP_CONT[_BF] primitives are decomposed identically to basic line strips, but the *continueStipple* variable is set to true so that the line stipple pattern will pick up from where it left off with the last line primitive, rather than being reset.

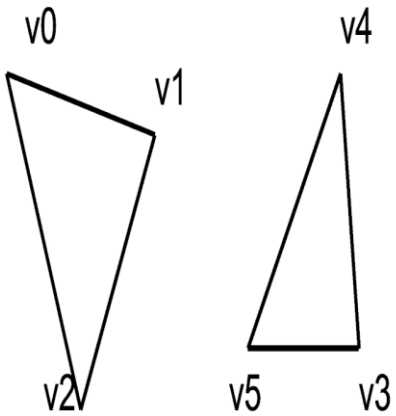
```
LineStripDecomposition()
   {
      objectType = 3DOBJ_LINE
      nV = 2
      provokingVtx = Line List/Strip Provoking Vertex Select
      if (primType == 3DPRIM_LINESTRIP)
         {
             backFacing = FALSE
             continueStipple = FALSE
         } else if (primType == 3DPRIM_LINESTRIP_BF)
            {
               backFacing = TRUE
               continueStipple = FALSE
            } else if (primType == 3DPRIM_LINESTRIP_CONT)
               {
                  backFacing = FALSE
                  continueStipple = TRUE
               } else if (primType == 3DPRIM_LINESTRIP_CONT_BF)
                  {
                     backFacing = TRUE
```

```
                    continueStipple = TRUE
                }
        for each (vertex I in [0..vertexCount-1])
            {
                v[0] arrays ← vIn[i] arrays
                v[1] arrays ← vIn[i+1] arrays
                ObjectSetup()
                continueStipple = TRUE
            }
    }
```

Triangle List Decomposition

The 3DPRIM_TRILIST primitive specifies a list of independent triangles.

3DPRIM_TRILIST Primitive



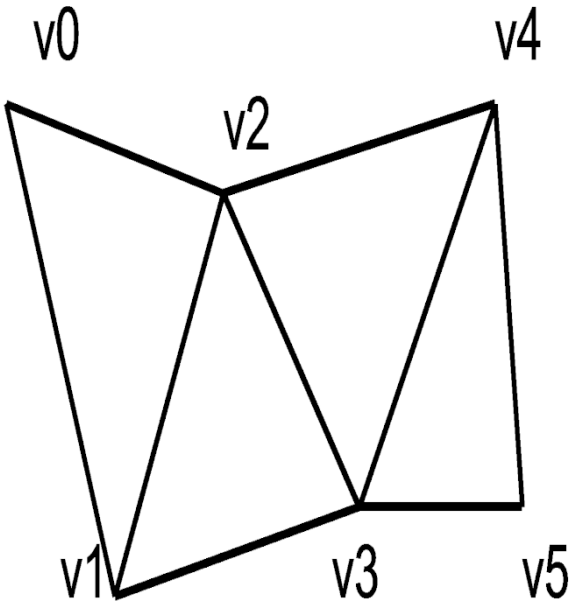
The decomposition process divides the list into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v3,v4,v5; and so on. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

```
TriangleListDecomposition() {
  objectType = 3DOBJ_TRIANGLE
      nV = 3
  invertOrientation = FALSE
  provokingVtx = Triangle List/Strip Provoking Vertex Select
  polyStippleEnable = TRUE
  for each (vertex I in [0..vertexCount-3] by 3)
      {
          v[0] arrays ← vIn[i] arrays
          v[1] arrays ← vIn[i+1] arrays
          v[2] arrays ← vIn[i+2] arrays
          ObjectSetup()
      }
}
```

Triangle Strip Decomposition

The 3DPRIM_TRISTRIP and 3DPRIM_TRISTRIP_REVERSE primitives specify a series of triangles arranged in a strip, as illustrated below.

3DPRIM_TRISTRIP[_REVERSE] Primitive



The decomposition process divides the strip into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v1,v2,v3; v2,v3,v4; and so on. Note that the *winding order* of the vertices alternates between CW (clockwise), CCW (counter-clockwise), CW, etc. The *provokingVertex* of each object is taken from the **Triangle List/Strip Provoking Vertex Select** state variable, as programmed via SF_STATE.

The 3D pipeline uses the winding order of the vertices to distinguish between front-facing and back-facing triangles (*Triangle Orientation (Face) Culling* below). Therefore, the 3D pipeline must account for the alternation of winding order in strip triangles. The *invertOrientation* variable is generated and used for this purpose.

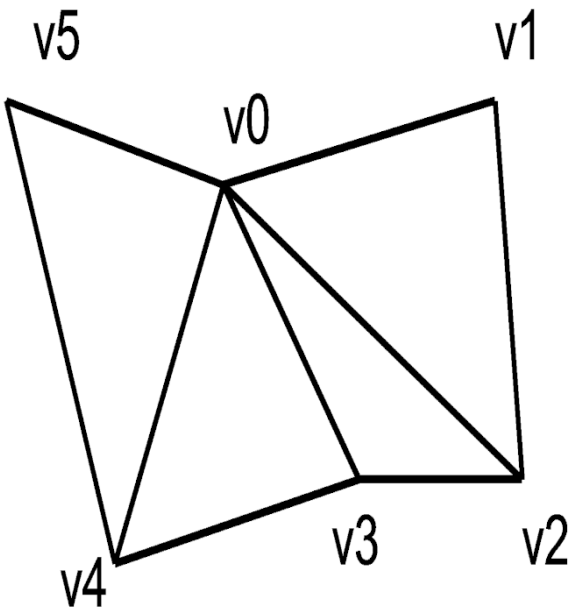
To accommodate the situation where the driver is forced to break an input strip primitive into multiple tristrip primitive commands (for example, due to ring or batch buffer size restrictions), two tristrip primitive types are supported. 3DPRIM_TRISTRIP is used for the initial section of a strip, and wherever a continuation of a strip starts with a triangle with a CW winding order. 3DPRIM_TRISTRIP_REVERSE is used for a continuation of a strip that starts with a triangle with a CCW winding order.

```
TriangleStripDecomposition()
  {
    objectType = 3DOBJ_TRIANGLE
    nV = 3
    provokingVtx = Triangle List/Strip Provoking Vertex Select
    if (primType == 3DPRIM_TRISTRIP)
      invertOrientation = FALSE
      else // primType == 3DPRIM_TRISTRIP_REVERSE
        invertOrientation = TRUE
    polyStippleEnable = TRUE
    for each (vertex I in [0..vertexCount-3])
     {
        v[0] arrays ← vIn[i] arrays
        v[1] arrays ← vIn[i+1] arrays
        v[2] arrays ← vIn[i+2] arrays
        ObjectSetup()
        invertOrientation = ! invertOrientation
     }
  }
```

Triangle Fan Decomposition

The 3DPRIM_TRIFAN and 3DPRIM_TRIFAN_NOSTIPPLE primitives specify a series of triangles arranged in a fan, as illustrated below.

3DPRIM_TRIFAN Primitive



The decomposition process divides the fan into a series of basic 3DOBJ_TRIANGLE objects that are then passed individually and in order to the Object Setup stage. The triangles are generated with the following object vertex order: v0,v1,v2; v0,v2,v3; v0,v3,v4; and so on. As there is no alternation in the vertex winding order, the *invertOrientation* variable is output as FALSE unconditionally. The *provokingVertex* of each object is taken from the **Triangle Fan Provoking Vertex** state variable, as programmed via SF_STATE.

Primitives of type 3DPRIM_TRIFAN_NOSTIPPLE are decomposed exactly the same way, except the *polyStippleEnable* variable is FALSE for the resulting objects being passed on to object setup. This will inhibit polygon stipple for these triangle objects.

```
TriangleFanDecomposition()
   {
       objectType = 3DOBJ_TRIANGLE
       nV = 3
       invertOrientation = FALSE
       provokingVtx = Triangle Fan Provoking Vertex Select
       if (primType == 3DPRIM_TRIFAN)
           polyStippleEnable = TRUE
           else // primType == 3DPRIM_TRIFAN_NOSTIPPLE
              polyStippleEnable = FALSE

       v[0] arrays ← vIn[0] arrays
         // the 1st vertex is common

       for each (vertex I in [1..vertexCount-2])
          {
              v[1] arrays ← vIn[i] arrays
              v[2] arrays ← vIn[i+1] arrays
              ObjectSetup()
          }
   }
```

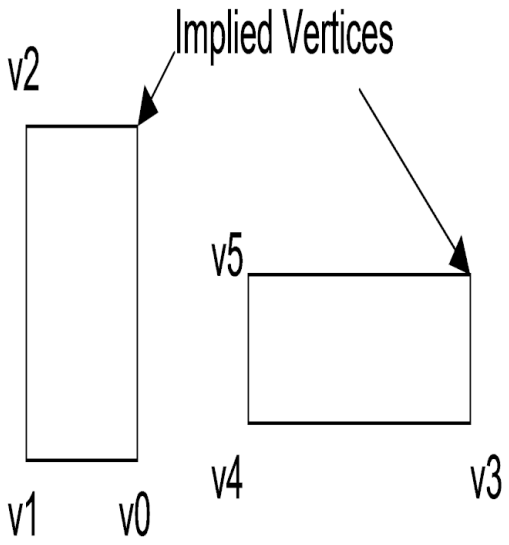
Polygon Decomposition

The 3DPRIM_POLYGON primitive is identical to the 3DPRIM_TRIFAN primitive with the exception that the *provokingVtx* is overridden with 0. This support has been added specifically for OpenGL support, avoiding the need for the driver to change the provoking vertex selection when switching between trifan and polygon primitives.

Rectangle List Decomposition

The 3DPRIM_RECTLIST primitive command specifies a list of independent, axis-aligned rectangles. Only the lower right, lower left, and upper left vertices (in that order) are included in the command – the upper right vertex is derived from the other vertices (in Object Setup).

3DPRIM_RECTLIST Primitive



The decomposition of the 3DPRIM_RECTLIST primitive is identical to the 3DPRIM_TRILIST decomposition, with the exception of the *objectType* variable.

```
RectangleListDecomposition()
   {
       objectType = 3DOBJ_RECTANGLE
       nV = 3
       invertOrientation = FALSE
       provokingVtx = 0
       for each (vertex I in [0..vertexCount-3] by 3)
          {
              v[0] arrays ←  vIn[i] arrays
              v[1] arrays ←  vIn[i+1] arrays
              v[2] arrays ←  vIn[i+2] arrays
              ObjectSetup()
          }
   }
```

Object Setup

The Object Setup subfunction of the SF stage takes the post-viewport-transform data associated with each vertex of a basic object and computes various parameters required for scan conversion. This includes generation of implied vertices, translations and adjustments on vertex positions, and culling (removal) of certain classes of objects. The final object information is passed to the Windower/Masker (WM) stage where the object is rasterized into pixels.

Invalid Position Culling (Pre/Post-Transform)

At input the the SF stage, any objects containing a floating-point NaN value for Position X, Y, Z, or RHW will be unconditionally discarded. Note that this occurs on an object (not primitive) basis.

If Viewport Transformation is enabled, any objects containing a floating-point NaN value for post-transform Position X, Y or Z will be unconditionally discarded.

Viewport Transformation

If the **Viewport Transform Enable** bit of SF_STATE is ENABLED, a viewport transformation is applied to each vertex of the object.

The VPIndex associated with the leading vertex of the object is used to obtain the **Viewport Matrix Element** data from the corresponding element of the SF_VIEWPORT structure in memory. For each object vertex, the following scale and translate transformation is applied to the position coordinates:

$x' =$ **m00** $* x +$ **m30**

$y' =$ **m11** $* y +$ **m31**

$z' =$ **m22** $* z +$ **m32**

Software is responsible for computing the matrix elements from the viewport information provided to it from the API.

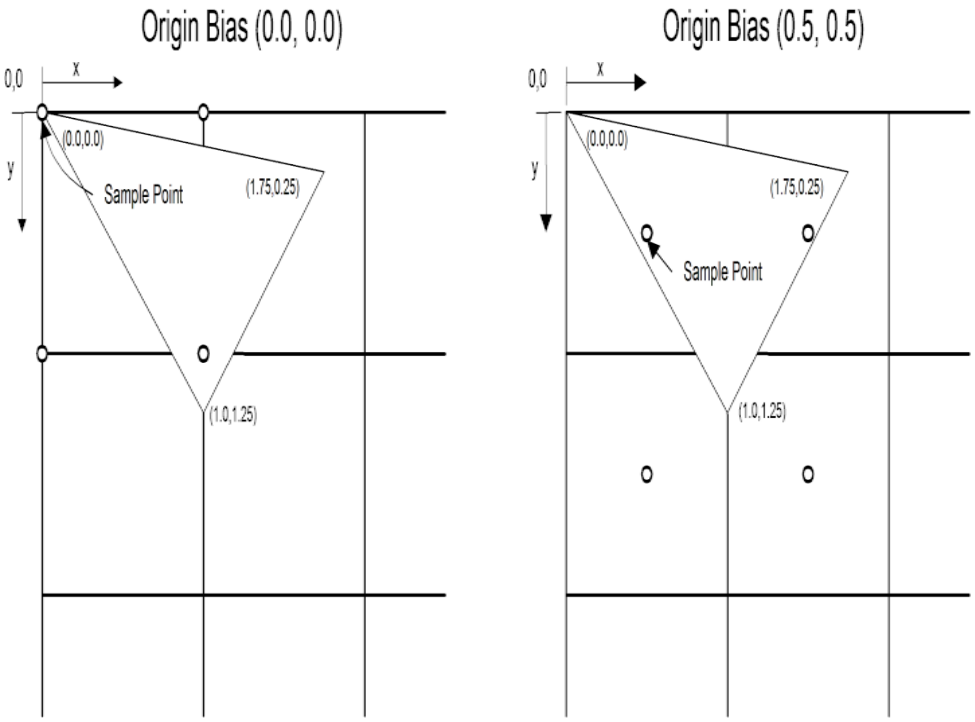
Destination Origin Bias

The positioning of the pixel sampling grid is programmable and is controlled by the **Destination Origin Horizontal/Vertical Bias** state variables (set via SF_STATE). If these bias values are both 0, pixels are sampled on an integer grid. Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0,0) falls within the primitive.

If the bias values are both 0.5, pixels are sampled on a “half” integer grid (i.e., X.5, Y.5). Pixel (0,0) will be considered inside the object if the sample point at XY coordinate (0.5,0.5) falls within the primitive. This positioning of the sample grid corresponds with the OpenGL rasterization rules, where “fragment centers” lay on a half-integer grid. It also corresponds with the Intel740 rasterizer (though that device did not employ “top left” rules).

Note that subsequent descriptions of rasterization rules for the various objects will be with reference to the pixel sampling grid.

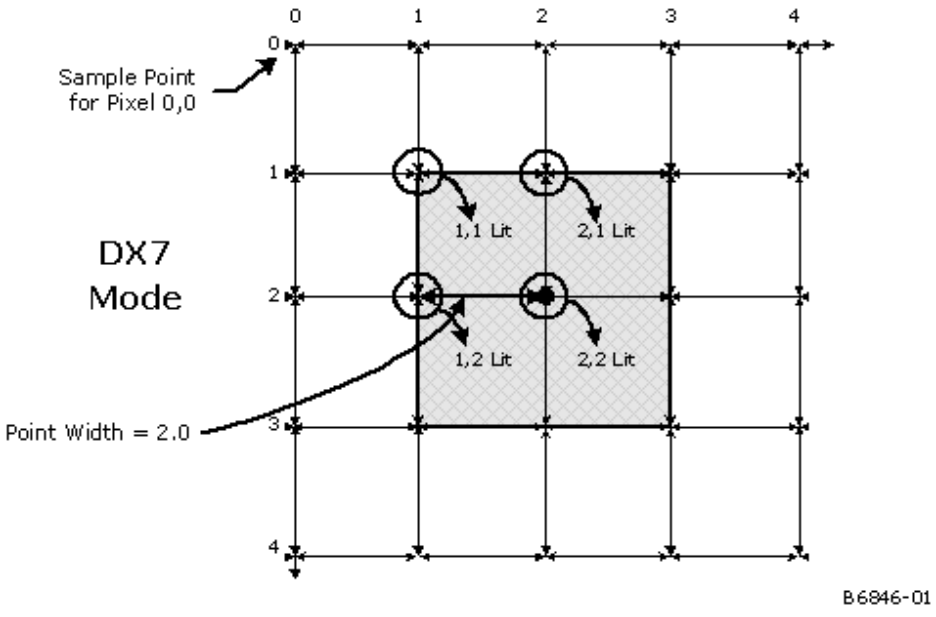
Destination Origin Bias



Point Rasterization Rule Adjustment

POINT objects are rasterized as square RECTANGLEs, with one exception: The **Point Rasterization Rule** state variable (in SF_STATE) controls the rendering of point object edges that fall directly on pixel sample points, as the treatment of these edge pixels varies between APIs.

RASTRULE_UPPER_LEFT

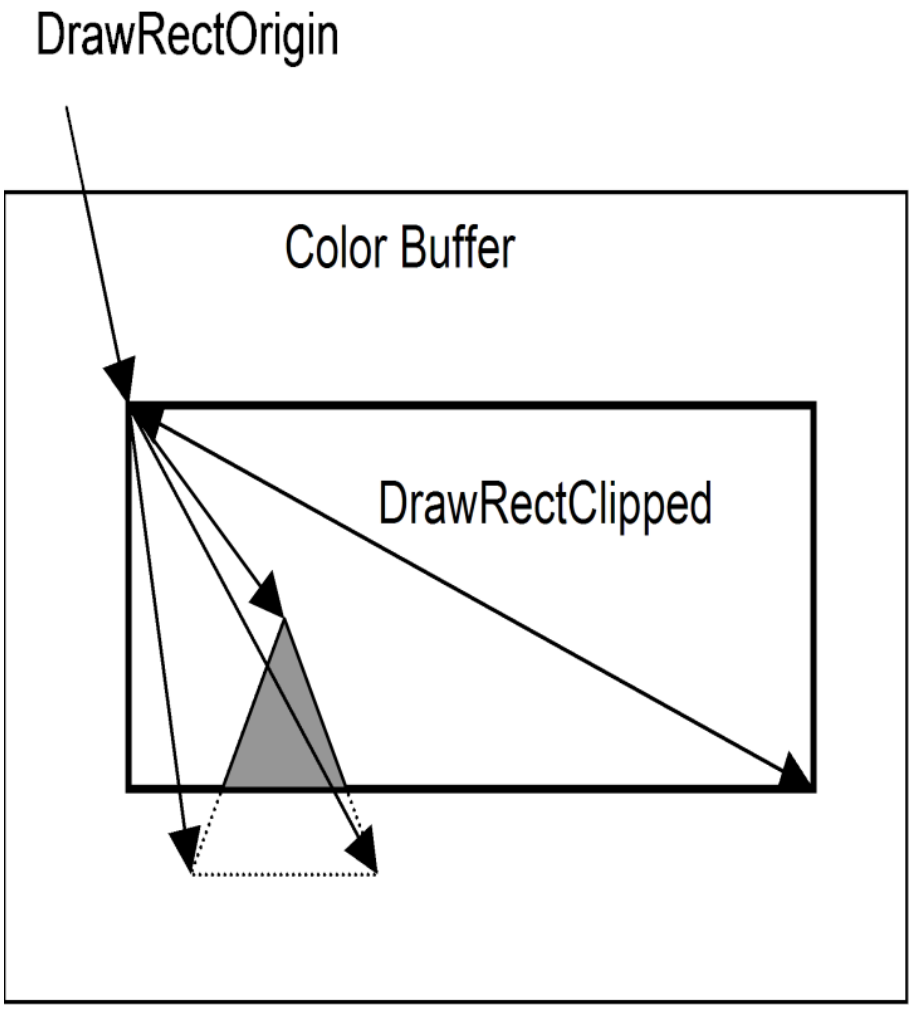


Drawing Rectangle Offset Application

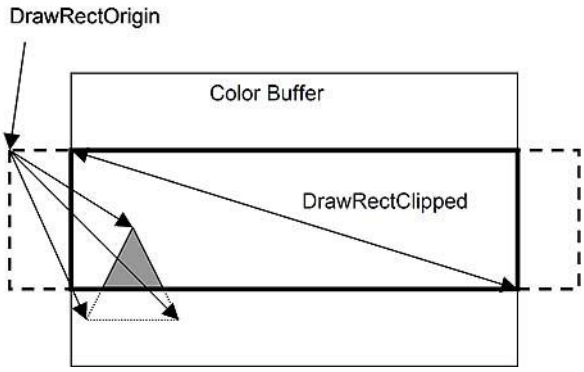
The Drawing Rectangle Offset subfunction offsets the object's vertex X,Y positions by the pixel-exact, unclipped drawing rectangle origin (as programmed via the **Drawing Rectangle Origin X,Y** values in the 3DSTATE_DRAWING_RECTANGLE command). The Drawing Rectangle Offset subfunction (at least with respect to Color Buffer access) is unconditional, and therefore to (effectively) turn off the offset function the origin would need to be set to (0,0). A non-zero offset is typically specified when window-relative or viewport-relative screen coordinates are input to the device. Here the drawing rectangle origin would be loaded with the absolute screen coordinates of the window's or viewport's upper-left corner.

Clipping of objects which extend outside of the Drawing Rectangle occurs later in the pipeline. Note that this clipping is based on the "clipped" draw rectangle (as programmed via the **Clipped Drawing Rectangle** values in the 3DSTATE_DRAWING_RECTANGLE command), which must be clamped by software to the rendertarget boundaries. The unclipped drawing rectangle origin, however, can extend outside the screen limits in order to support windows whose origins are moved off-screen. This is illustrated in the following diagrams.

Onscreen Draw Rectangle



Partially-offscreen Draw Rectangle



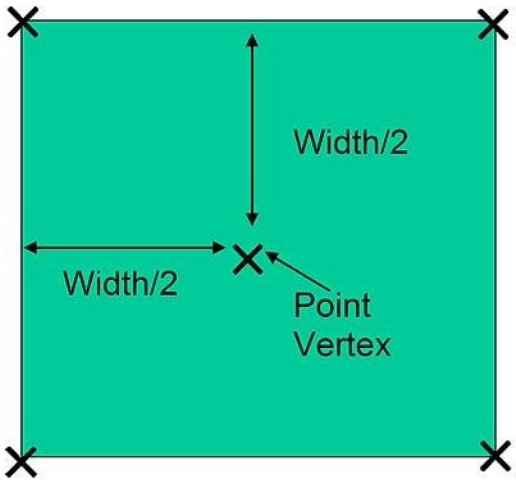
3DSTATE_DRAWING_RECTANGLE

Point Width Application

This stage of the pipeline applies only to 3DOBJ_POINT objects. Here the point object is converted from a single vertex to four vertices located at the corners of a square centered at the point's X,Y position. The width and height of the square are specified by a *point width* parameter. The **Point Width Source** value in SF_STATE determines the source of the point width parameter: the point width is either taken from the **Point Width** value programmed in SF_STATE or the PointWidth specified with the vertex (as read back from the vertex VUE earlier in the pipeline).

The corner vertices are computed by adding and subtracting one half of the point width. *Point Width Application*.

Point Width Application

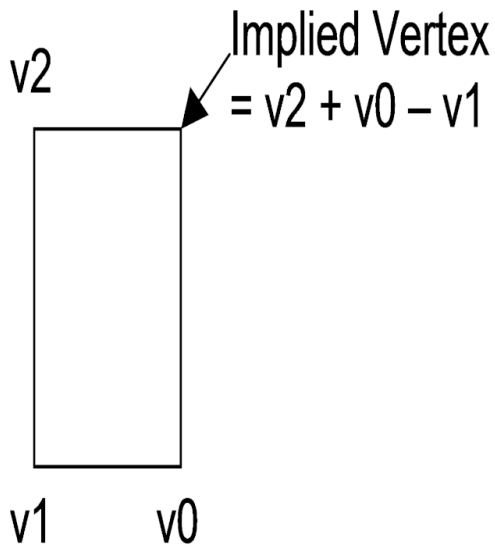


Z and W vertex attributes are copied from the single point center vertex to each of the four corner vertices.

Rectangle Completion

This stage of the pipeline applies only to 3DOBJ_RECTANGLE objects. Here the X,Y coordinates of the 4th (upper right) vertex of the rectangle object is computed from the first 3 vertices as shown in the following diagram. The other vertex attributes assigned to the implied vertex (v[3]) are UNDEFINED as they are not used. The Object Setup subfunction will use the values at only the first 3 vertices to compute attribute interpolants used across the entire rectangle.

Rectangle Completion



Vertex XY Clamping and Quantization

At this stage of the pipeline, vertex X and Y positions are in continuous screen (pixel) coordinates. These positions are quantized to subpixel precision by rounding the incoming values to the nearest subpixel (using round-to-nearest-or-even rules matching the DirectX reference device). The device supports rasterization with either 4 or 8 fractional (subpixel) position bits, as specified by the **Vertex SubPixel Precision Select** bit of SF_STATE.

The vertex X and Y screenspace coordinates are also *clamped* to the fixed-point "guardband" range supported by the rasterization hardware, as listed in the following table:

Per-Device Guardband Extents

Supported X,Y ScreenSpace "Guardband" Extent	Maximum Post-Clamp Delta (X or Y)
[-32K,32K-1]	N/A

Note that this clamping occurs after the Drawing Rectangle Origin has been applied and objects have been expanded (i.e., points have been expanded to squares, etc.). In almost all circumstances, if an object's vertices are actually modified by this clamping (i.e., had X or Y coordinates outside of the guardband extent the rendered object will not match the intended result. Therefore software should take steps to ensure that this does not happen – e.g., by clipping objects such that they do not exceed these limits after the Drawing Rectangle is applied.

In addition, in order to be correctly rendered, objects must have a screenspace bounding box not exceeding 8K in the X or Y direction. This additional restriction must also be comprehended by software, i.e., enforced by use of clipping.

Degenerate Object Culling

At this stage of the pipeline, "degenerate" objects are discarded. This operation is automatic and cannot be disabled. (The object rasterization rules would by definition cause these objects to be "invisible" – this culling operation is mentioned here to reinforce that the device implementation optimizes these degeneracies as early as possible).

Degenerate Object Culling for definitions of degenerate objects.

Degenerate Objects

objType	Degenerate Object Definition
3DOBJ_POINT	Two or more corner vertices are coincident (i.e., the radius quantized to zero)
3DOBJ_LINE	The endpoints are coincident
3DOBJ_TRIANGLE	All three vertices are collinear or any two vertices are coincident and SOLID fill mode applies to the triangle
3DOBJ_RECTANGLE	Two or more corner vertices are coincident

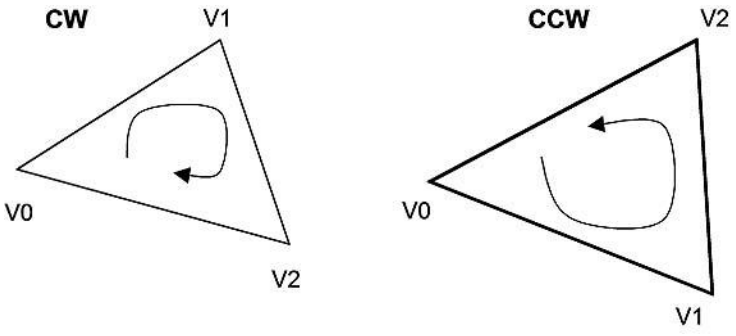
Triangle Orientation (Face) Culling

At this stage of the pipeline, 3DOBJ_TRIANGLE objects can be optionally discarded based on the "face orientation" of the object. This culling operation does not apply to the other object types.

This operation is typically called "back face culling", though front facing objects (or all 3DOBJ_TRIANGLE objects) can be selected to be discarded as well. Face culling is typically used to eliminate triangles facing away from the viewer, thus reducing rendering time.

The "winding order" of a triangle is defined by the the triangle vertex's 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal proceeds in either a clockwise (CW) or counter-clockwise (CCW) direction. The "winding order" of a triangle is defined by the the triangle vertex's 2D (X,Y) screen space position when traversed from v0 to v1 to v2. That traversal will proceed in either a clockwise (CW) or counter-clockwise (CCW) direction. A degenerate triangle is considered "backfacing", regardless of the FrontWinding state.

Triangle Winding Order



The **Front Winding** state variable in SF_STATE controls whether CW or CCW triangles are considered as having a "front-facing" orientation (at which point non-front-facing triangles are considered "back-

facing"). The internal variable *invertOrientation* associated with the triangle object is then used to determine whether the orientation of a that triangle should be inverted. Recall that this variable is set in the Primitive Decomposition stage to account for the alternating orientations of triangles in strip primitives resulting form the ordering of the vertices used to process them.

The **Cull Mode** state variable in SF_STATE specifies how triangles are discarded according to their resultant orientation. See Degenerate Objects.

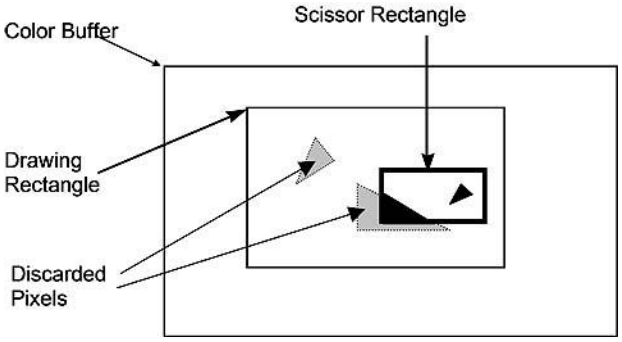
Cull Mode

CullMode	Definition
CULLMODE_NONE	The face culling operation is disabled.
CULLMODE_FRONT	Triangles with "front facing" orientation are discarded.
CULLMODE_BACK	Triangles with "back facing" orientation are discarded.
CULLMODE_BOTH	All triangles are discarded.

Scissor Rectangle Clipping

A *scissor* operation can be used to restrict the extent of rendered pixels to a screen-space aligned rectangle. If the scissor operation is enabled, portions of objects falling outside of the intersection of the scissor rectangle and the clipped draw rectangle are clipped (pixels discarded).

The scissor operation is enabled by the **Scissor Rectangle Enable** state variable in SF_STATE. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding SF_VIEWPORT structure. Up to 16 structures are supported. The **Scissor Rectangle X,Y Min,Max** fields of the SF_VIEWPORT structure defines a scissor rectangle as a rectangle in integer pixel coordinates relative to the (unclipped) origin of the Drawing Rectangle. The scissor rectangle is defined relative to the Drawing Rectangle to better support the OpenGL API. (OpenGL specifies the "Scissor Box" in window-relative coordinates). This allows instruction buffers with embedded Scissor Rectangle definitions to remain valid even after the destination window (drawing rectangle) moves.



Specifying either scissor rectangle xmin > xmax or ymin > ymax will cause all polygons to be discarded for a given viewport (effectively a null scissor rectangle).

Viewport Extents Test

Viewport extents test can be used to restrict the extent of rendered pixels to the viewport extents. If this operation is enabled, portion of the objects falling outside of the intersection of the scissor rectangle (if enabled) and the clipped draw rectangle and viewport extents are clipped (pixels discarded). This operation similar to the scissor test except both have different enables and the viewport extents can be programmed to the fractional float values.

This operation is enabled by the **View Transform Enable** state variable in **SF_STATE**. If enabled, the VPIndex associated with the leading vertex of the object is used to select the corresponding **SF_CLIP_VIEWPORT** structure. Up to 16 structures are supported. The X/Y **Min/Max ViewPort** fields of the **SF_CLIP_VIEWPORT** structure defines viewport extents as a rectangle in float screen pixel coordinates relative to the (unclipped) origin of the Drawing Rectangle. Please note that these co-ordinates can be fractional values and hardware will do appropriate rounding and convert it to integer pixel co-ordinates. This **View Transform Enable** state also controls the viewport transform so appropriate the viewport transform coefficients need to be populated in the SF_CLIP_VIEPWORT structure along with the viewport extents.

Final clip rectangle used to define the rendering area will now depend on three rectangles namely drawing rectangle, Scissor rectangle, Viewport Extents. If both **Scissor Rectangle Enable** and **View transform enable** are set then intersection of all rectangles (Viewport extents, Scissor rectangle, Draw rectangle) becomes final clip rectangle, while If only **Scissor Rectangle Enable** is enabled then the intersection of (Scissor rectangle, Draw rectangle) becomes final clip rectangle. If only **View transform enable** is enabled then intersection of (Viewport extents, Draw rectangle) become the final clip rectangle, while If none is enabled then (Draw rectangle) is the final clip rectangle.

Line Rasterization

The device supports three styles of line rendering: *zero-width* (*cosmetic*) lines, *non-antialiased* lines, and *antialiased* lines. Non-antialiased lines are rendered as a polygon having a specified width as measured parallel to the major axis of the line. Antialiased lines are rendered as a rectangle having a specified width measured perpendicular to the line connecting the vertices.

The functions required to render lines are split between the SF and WM units. The SF unit is responsible for computing the overall geometry of the object to be rendered, including the pixel-exact bounding box, edge equations, etc., and therefore is provided with the screen-geometry-related state variables. The WM unit performs the actual scan conversion, determining the exact pixels included/excluded and coverage values for anti-aliased lines.

Zero-Width (Cosmetic) Line Rasterization

Note: The specification of zero-width line rasterization would be more correctly included in the WM Unit chapter, though is being included here to keep it with the rasterization details of the other line types.

When the **Line Width** is set to zero, the device will use special rules to rasterize zero-width (“cosmetic”) lines. The **Anti-Aliasing Enable** state variable is ignored when **Line Width** is zero.

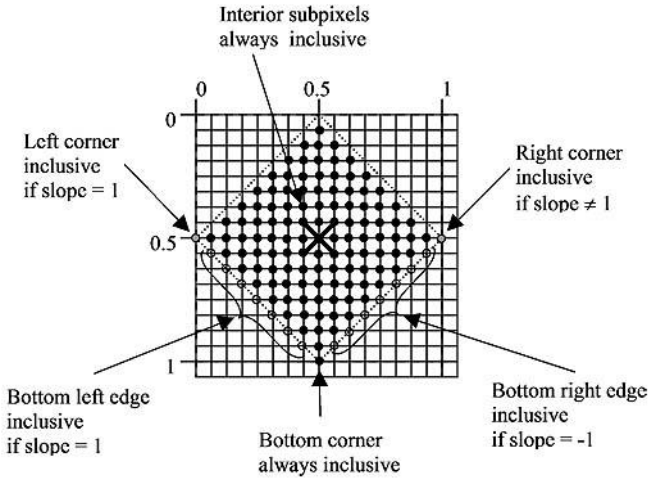
When the **LineWidth** is set to zero, the device will use special rules to rasterize "cosmetic" lines. The rasterization rules also comply with the OpenGL conformance requirements (for 1-pixel wide non-smooth lines). Refer to the appropriate API specifications for details on these requirements.

The GIQ rules basically intersect the directed, ideal line connecting two endpoints with an array of diamond-shaped areas surrounding pixel sample points. Wherever the line exits a diamond (including passing through a diamond), the corresponding pixel is lit. Special rules are used to define the subpixel locations that are considered interior to the diamonds, as a function of the slope of the line. When a line ends in a diamond (and therefore does not exit that diamond), the corresponding pixel is not drawn. When a line starts in a diamond and exits that diamond, the corresponding pixel is drawn.

GIQ (Diamond) Sampling Rules – Legacy Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is ENABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered "interior" to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line slope is not exactly one, in which case the left corner subpixel is inclusive.** Including the right corner subpixel ensures that lines with slopes in the range (1, +infinity] or [-infinity, -1) touch a diamond even when they cross exactly between pixel diamonds. Including the left corner on slope=1 lines is required for proper

handling of slope=1 lines (see (3) below) – where if the right corner was inclusive, a slope=1 line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).

3. **The subpixels along the bottom left edge are inclusive only if the line slope = 1.** This is to correctly handle the case where a slope=1 line falls enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this "passing through" the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One slope=1 line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would case the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as "inside" for slope=1 lines, we will only light the pixel when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.
4. **The subpixels along the bottom right edge are inclusive only if the line slope = -1.** Similar case as (3), except slope=-1 lines require the bottom right edge to be considered inclusive.

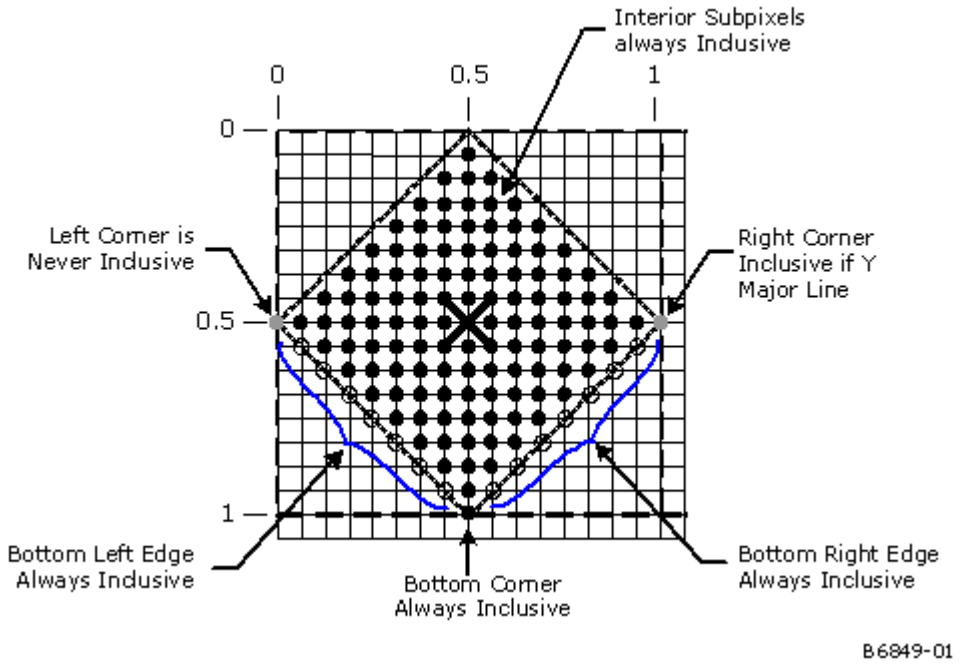
The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (slopePosOne, slopeNegOne).

```
delta_x                         = point.x - sample.x
delta_y                         = point.y - sample.y
distance                = abs(delta_x) + abs(delta_y)
interior                = (distance < 0.5)
bottom_corner           = (delta_x == 0.0) && (delta_y == 0.5)
left_corner             = (delta_x == -0.5) && (delta_y == 0.0)
right_corner            = (delta_x == 0.5) && (delta_y == 0.0)
bottom_left_edge        = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)
bottom_right_edge       = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior || bottom_corner || (slopePosOne ? left_corner : right_corner) ||
(slopePosOne && left_edge) || (slopeNegOne && right_edge)
```

GIQ (Diamond) Sampling Rules – DX10 Mode

When the **Legacy Line Rasterization Enable** bit in WM_STATE is DISABLED, zero-width lines are rasterized according to the algorithm presented in this subsection. Also note that the **Last Pixel Enable** bit of SF_STATE controls whether the last pixel of the last line in a LINESTRIP_xxx primitive or the last pixel of each line in a LINELIST_xxx primitive is rendered.

Refer to the following figure, which shows the neighborhood of subpixels around a given pixel sample point. Note that the device divides a pixel into a 16x16 array of subpixels, referenced by their upper left corners.



The solid-colored subpixels are considered "interior" to the diamond centered on the pixel sample point. Here the Manhattan distance to the pixel sample point (center) is less than ½.

The subpixels falling on the edges of the diamond (Manhattan distance = ½) are exclusive, with the following exceptions:

1. **The bottom corner subpixel is always inclusive.** This is to ensure that lines with slopes in the open range (-1,1) touch a diamond even when they cross exactly between pixel diamonds.
2. **The right corner subpixel is inclusive as long as the line is not X Major (X Major is defined as -1 <= slope <= 1).** Including the right corner subpixel ensures that lines with slopes in the range (>1, +infinity] or [-infinity, <-1) touch a diamond even when they cross exactly between pixel diamonds.
3. **The left corner subpixel is never inclusive.** For Y Major lines, having the right corner subpixel as always inclusive requires that the left corner subpixel should never be inclusive, since a line falling exactly between pixel centers would wind up lighting pixel on both sides of the line (not desired).
4. **The subpixels along the bottom left edge are always inclusive.** This is to correctly handle the case where a line enters the diamond through a left or bottom corner and ends on the bottom left edge. One does not consider this "passing through" the diamond (where the normal rules would have us light the pixel). This is to avoid the following case: One line segment enters through one corner and ends on the edge, and another (continuation) line segments starts at that point on the edge and exits through the other corner. If simply passing through a corner caused the pixel to be lit, this case would cause the pixel to be lit twice – breaking the rule that connected line segments should not cause double-hits or missing pixels. So, by considering the entire bottom left edge as "inside", the pixel is only lit when a line passes through the entire edge, or starts on the edge (or the left or bottom corner) and exits the diamond.

5. **The subpixels along the bottom right edge are always inclusive.** Same as case as (4), except slope=-1 lines require the bottom right edge to be considered inclusive.

The following equation determines whether a point (point.x, point.y) is inside the diamond of the pixel sample point (sample.x, sample.y), given additional information about the slope (XMajor).

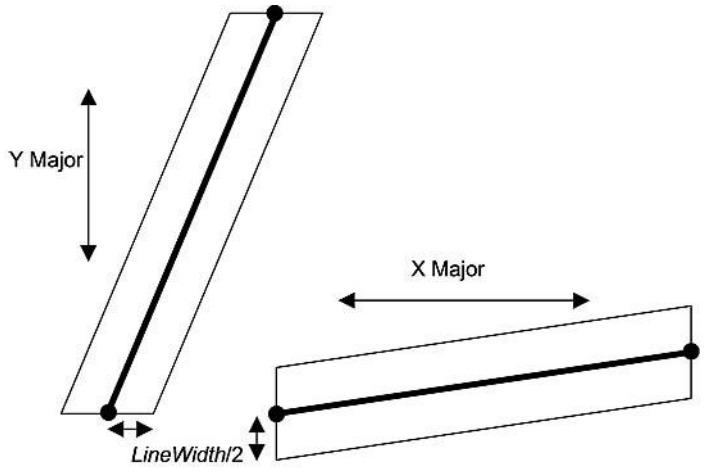
```
delta_x                         = point.x - sample.x
delta_y                         = point.y - sample.y
distance                        = abs(delta_x) + abs(delta_y)
interior                        = (distance < 0.5)
bottom_corner                           = (delta_x == 0.0)  && (delta_y == 0.5)
left_corner                     = (delta_x == -0.5) && (delta_y == 0.0)
right_corner                    = (delta_x == 0.5)  && (delta_y == 0.0)
bottom_left_edge                = (distance == 0.5) && (delta_x < 0) && (delta_y > 0)
bottom_right_edge               = (distance == 0.5) && (delta_x > 0) && (delta_y > 0)

inside = interior || bottom_corner || (!XMajor && right_corner) || ( bottom_left_edge)
|| ( bottom_right_edge)
```

Non-Antialiased Wide Line Rasterization

Non-anti-aliased, non-zero-width lines are rendered as parallelograms that are centered on, and aligned to, the line joining the endpoint vertices. Pixels sampled interior to the parallelogram are rendered; pixels sampled exactly on the parallelogram edges are rendered according to the polygon "top left" rules.

The parallelogram is formed by first determining the major axis of the line (diagonal lines are considered x-major). The corners of the parallelogram are computed by translating the line endpoints by +/-(**Line Width** / 2) in the direction of the minor axis, as shown in the following diagram.

Non-Antialiased Line Rasterization

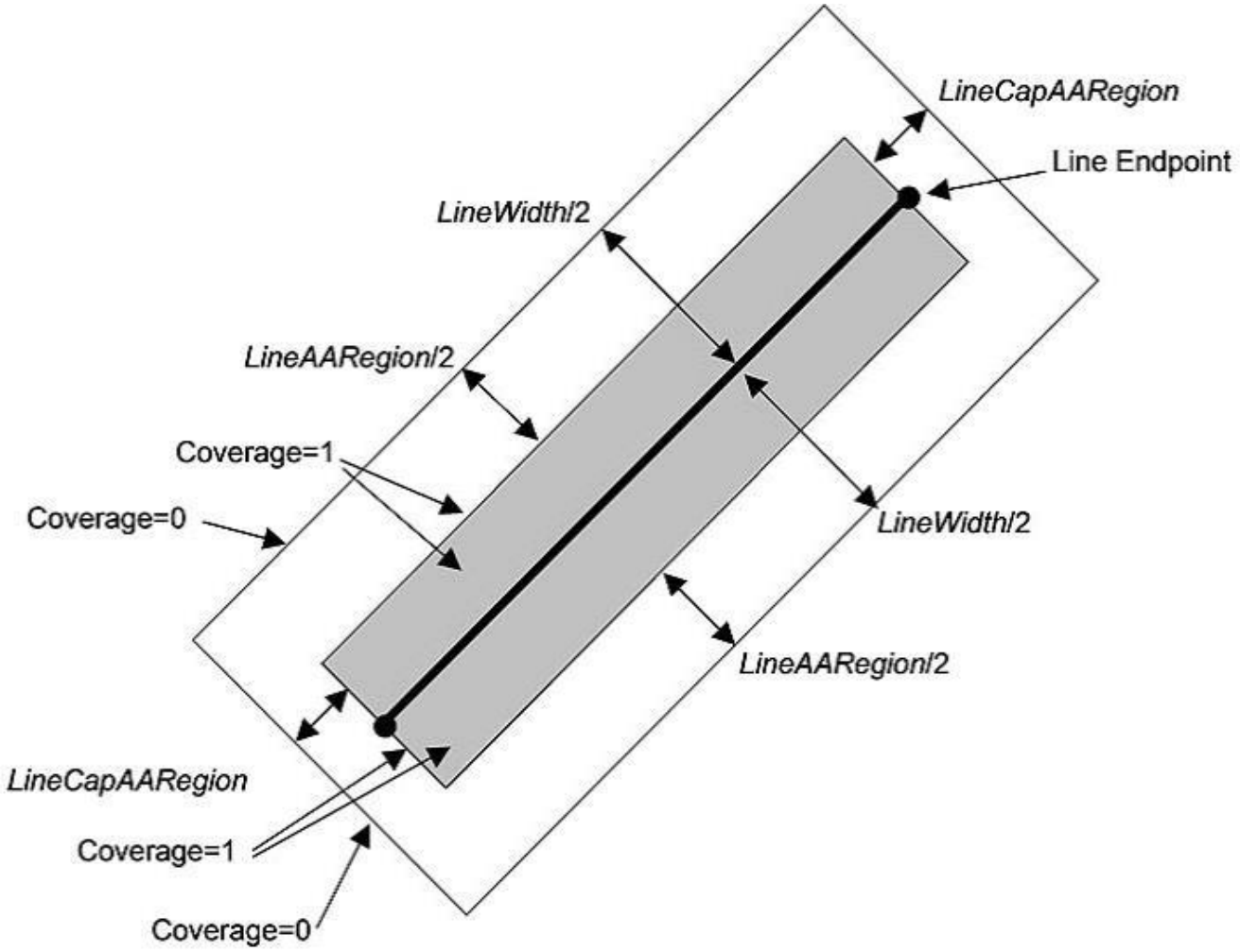


Anti-Aliased Line Rasterization

Anti-aliased lines are rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices. For each pixel in the rectangle, a fractional coverage value (referred to as Antialias Alpha) is computed – this coverage value is normally used to attenuate the pixel's alpha in the pixel shader thread. The resultant alpha value is therefore available for use in those downstream pixel pipeline stages to generate the desired effect (e.g., use the attenuated alpha value to modulate the pixel's color, and add the result to the destination color, etc.). Note that software is required to explicitly program the pixel shader and pixel pipeline to obtain the desired anti-aliasing effect – the device simply makes the coverage-attenuated pixel alpha values available for use in the pixel shader.

The dimensions of the rendered rectangle, and the parameters controlling the coverage value computation, are programmed via the **Line Width**, **Line AA Region**, and **Line Cap AA Region** state variables, as shown below. The edges parallel to the line are located at the distance (*LineWidth*/2) from the line (measured in screen pixel units perpendicular to the line). The end-cap edges are perpendicular to the line and located at the distance (*LineCapAARegion*) from the endpoints.

Anti-aliased Line Rasterization



Along the parallel edges, the coverage values ramp from the value 0 at the very edges of the rectangle to the value 1 at the perpendicular distance (*LineAARegion*/2) from a given edge (in the direction of the line). A pixel's coverage value is computed with respect to the closest edge. In the cases where (*LineAARegion*/2) < (*LineWidth*/2), this results in a region of fractional coverage values near the edges of the rectangle, and a region of "fully-covered" coverage values (i.e., the value 1) at the interior of the line. When (*LineAARegion*/2) == (*LineWidth*/2), only pixel sample points falling exactly on the line can generate fully-covered coverage values. If (*LineAARegion*/2) > (*LineWidth*/2), no pixels can be fully-covered (it is expected that this case is not typically desired).

Along the end cap edges, the coverage values ramp from the value 1 at the line endpoint to the value 0 at the cap edge – itself at a perpendicular distance (*LineCapAARegion*) from the endpoint. Note that, unlike the line-parallel edges, there is only a single parameter (*LineCapAARegion*) controlling the extension of the line at the end caps and the associated coverage ramp.

The regions near the corners of the rectangle have coverage values influenced by distances from both the line-parallel and end cap edges – here the two coverage values are multiplied together to provide a composite coverage value.

The computed coverage value for each pixel is passed through the Windower Thread Dispatch payload. The Pixel Shader kernel should be passed (unmodified) by the shader to the Render Cache as part of it's output message.

SF Pipeline State Summary

3DSTATE_RASTER [CHV, BSW]

3DSTATE_RASTER

Signal	SF_INT::Multisample Rasterization Mode
Description	This field determines whether multisample rasterization is enabled and how pixel sampl
Formula	See Table: WM_INT::Multisample Rasterization Mode in 3D Pipeline Windower [CHV, BS > 3DSTATE_WM > 3DSTATE_WM [CHV, BSW]

3DSTATE_SF

The state used by the SF stage is defined by this inline state packet.

3DSTATE_SF

The SF Unit also receives 3DSTATE_RASTER. It also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_WM, 3DSTATE_WM_HZ_OP, 3DSTATE_DETPH_BUFFER, and 3DSTATE_MULTISAMPLE fields.

Signal [CHV, BSW]	SF_INT::Number of Multisamples
Description	Set the number of multisamples.
Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP::Number of Multisamples : 3DSTATE_MULTISAMPLE::Number of Multisamples

Signal [CHV, BSW]	SF_INT::Pixel Position Offset Enable
Description	Enables the device to offset pixel positions by 0.5 both in horizontal and vertical directions.
Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP:: Pixel Position Offset Enable: 3DSTATE_MULTISAMPLE:: Pixel Position Offset Enable

Signal [CHV, BSW]	SF_INT::Pixel Position Offset
Description	Causes the device to offset pixel positions by 0.5 both in horizontal and vertical directions. It is to be noted this is done to adjust the pixel co-ordinate system to DX9 like, so any screen space rectangles (eg: HiZ Clear, Resolve etc) generated internally by driver in this mode needs to be aware of this offset adjustment and send the rectangles according to alignment restriction taking this offset adjustment into consideration.
Formula	= (SF_INT::Number of Multisamples >1) && (3DSTATE_MULTISAMPLE:: Pixel Location == PIXLOC_UL_CORNER) && SF_INT::Pixel Position Offset Enable

Signal [CHV, BSW]	SF_INT::Global Depth Offset Enable Solid
Description	Enables computation and application of Global Depth Offset for SOLID objects.
Formula	= 3DSTATE_RASTER:: Global Depth Offset Enable Solid

Signal	SF_INT::Global Depth Offset Enable Wireframe

Description	Enables computation and application of Global Depth Offset when triangles are rendered in WIREFRAME mo
Formula	= 3DSTATE_RASTER:: Global Depth Offset Enable Wireframe

Signal [CHV, BSW]	SF_INT::Global Depth Offset Enable Point
Description	Enables computation and application of Global Depth Offset when triangles are rendered in POINT mode.
Formula	= 3DSTATE_RASTER::Global Depth Offset Enable Point

Signal [CHV, BSW]	SF_INT::FrontFace Fill Mode
Description	This state controls how front-facing triangle and rectangle objects are rendered.
Formula	= 3DSTATE_INT::WM_HZ_OP ? SOLID : 3DSTATE_RASTER:: FrontFace Fill Mode

Signal [CHV, BSW]	SF_INT::BackFace Fill Mode
Description	This state controls how Back-facing triangle and rectangle objects are rendered.
Formula	= 3DSTATE_INT::WM_HZ_OP ? SOLID : 3DSTATE_RASTER:: BackFace Fill Mode

Signal [CHV, BSW]	SF_INT::FrontWinding
Description	Determines whether a triangle object is considered “front facing” if the screen space vertex positions, when traversed in the order, result in a clockwise (CW) or counter-clockwise (CCW) winding order. Does not apply to points or lines.
Formula	= 3DSTATE_INT::WM_HZ_OP ? FRONTWINDING_CW : 3DSTATE_RASTER::FrontWinding

Signal [CHV, BSW]	SF_INT::Cull Mode
Description	Controls removal (culling) of triangle objects based on orientation. The cull mode only applies to triangle objects and does not apply to lines, points or rectangles.
Formula	= SF_INT::WM_HZ_OP ? CULLMODE_BACK :

	3DSTATE_RASTER:: Cull Mode

Signal [CHV, BSW]	SF_INT::Scissor Rectangle Enable
Description	This field is used to control whether the Viewport Z extents (near, far) are considered in VertexClipTest.
Formula	= SF_INT::WM_HZ_OP ? 3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable : 3DSTATE_RASTER::Scissor Rectangle Enable

Signal [CHV, BSW]	SF_INT::Anti-aliasing Enable
Description	This field enables “alpha-based” line antialiasing.
Formula	= = SF_INT::WM_HZ_OP ? 3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable : 3DSTATE_RASTER::Anti-aliasing Enable

Signal [CHV, BSW]	SF_INT::Global Depth Offset Constant
Description	Specifies the constant term in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Constant

Signal [CHV, BSW]	SF_INT::Global Depth Offset Scale
Description	Specifies the constant term in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Scale

Signal [CHV, BSW]	SF_INT::Global Depth Offset Clamp
Description	Specifies the clamp term used in the Global Depth Offset function.
Formula	= 3DSTATE_RASTER::Global Depth Offset Clamp

Signal [CHV, BSW]	SF_INT::Line Stipple Enable
Description	Specifies the clamp term used in the Global Depth Offset function.
Formula	= 3DSTATE_WM::Line Stipple Enable

Signal [CHV, BSW]	SF_INT::RT Independent Rasterization Enable
Description	Enables RT Independent Rasterization.

Formula	= 3DSTATE_INT::WM_HZ_OP ? Disable : 3DSTATE_RASTER::ForcedSampleCount != NUMRASTSAMPLES_0

Signal [CHV, BSW]	SF_INT::WM_HZ_OP
Description	Enables WM_HZ_OP.
Formula	= (3DSTATE_WM_HZ_OP::DepthBufferClear || 3DSTATE_WM_HZ_OP::DepthBufferResolve || 3DSTATE_WM_HZ_OP::Hierarchical Depth Buffer Resolve Enable || 3DSTATE_WM_HZ_OP::StencilBufferClear) ? Enable : Disable

Signal [CHV, BSW]	SF_INT:: View Transform Enable
Description	Enables View Transform
Formula	= SF_INT::WM_HZ_OP ? Disable : 3DSTATE_SF::View Transform Enable

Signal [CHV, BSW]	SF_INT::Render Target Array index
Description	Render Target Array index being render to
Formula	= 3DSTATE_INT::WM_HZ_OP ? 0 : Render Target Array index pipelined from clipper

Signal [CHV, BSW]	SF_INT::Depth Buffer Surface Format
Description	Depth format being used
Formula	= 3DSTATE_INT:: Depth Buffer Surface Format

Signal [CHV, BSW]	SF_INT::Viewport index
Description	Viewport being used.
Formula	= SF_INT::WM_HZ_OP ? 0 : Viewport index pipelined from clipper

SF_CLIP_VIEWPORT

The viewport-specific state used by both the SF and CL units (SF_CLIP_VIEWPORT) is stored as an array of up to 16 elements, each of which contains the DWords described below. The start of each element is spaced 16 DWords apart. The location of the first element of the array, as specified by both **Pointer to SF_VIEWPORT** and **Pointer to CLIP_VIEWPORT**, is aligned to a 64-byte boundary.

SCISSOR_RECT

Attribute Interpolation Setup

With the attribute interpolation setup function being implemented in hardware for [CHV, BSW], a number of state fields in 3DSTATE_SF are utilized to control interpolation setup.

Number of SF Output Attributes sets the number of attributes that will be output from the SF stage, not including position. This can be used to specify up to 32, and may differ from the number of input attributes. The number of input attributes is derived from the **Vertex URB Entry Read Length** field. Note that this field is also used to specify whether swizzling is to be performed on Attributes 0-15 or Attributes 16-32. See the state field definition for details.

Attribute Swizzling

The first or last set of 16 attributes can be swizzled according to certain state fields. **Attribute Swizzle Enable** enables the swizzling for all 16 of these attributes, and each of the attributes has a 2-bit **Swizzle Select** field that controls swizzling with the following settings:

- INPUTATTR – This attribute is sourced from AttrInputReg[SourceAttribute].
- INPUTATTR_FACING – This attribute is sourced from AttrInputReg[SourceAttribute] if the object is front-facing, otherwise it is sourced from AttrInputReg[SourceAttribute+1].
- INPUTATTR_W – This attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination).
- INPUTATTR_FACING – If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute]. WYZW (the W component of the source is copied to the X component of the destination). If the object is front-facing, this attribute is sourced from AttrInputReg[SourceAttribute+1]. WYZW.

Each of the first or last set of 16 attributes also has a 5-bit **Source Attribute** field which specify, per output attribute (not component), which input attribute sources the output attribute when INPUTATTR is selected for **Swizzle Select**. A **Source Attribute** value of 0 corresponds to the 128-bit attribute immediately following the vertex 4D position. If INPUTATTR_FACING is selected, this specifies the first of two consecutive (front,back) input attributes, where the SourceAttribute value can be an odd or even number (just not 31, as that would place the back-face input attribute past the end of the input max complement of input attributes).

Constant overriding is also available for the first or last set of 16 attributes. Each attribute has a **Constant Source** field which specifies the constant values per swizzled attribute, with the following settings available:

- XYZW = 0000
- XYZW = 0001
- XYZW = 1111

Each channel of each attribute has a **Component Override** field to control whether the corresponding channel is overridden with the constant value defined in **Constant Source**.

Interpolation Modes

All 32 attributes have a **Constant Interpolation Enable** state field bit to specify whether all components of the post-swizzled attribute are to be interpolated as constant values (not varying over the pixels of the object). If set, the attribute at the provoking vertex is copied to a0, and a1 and a2 are set to zero – this results in a constant interpolation of the provoking vertex value. If clear, the attribute is linearly interpolated. Attributes 0-15 are further subjected to Wrap Shortest processing on a per-component basis, via the **Attribute WrapShortest Enables** state bitfields. WrapShortest processing modifies the a1 and/or a2 values depending on attribute deltas. All

The table below indicates the output values of a0, a1, and a2 depending on interpolation mode settings.

	a0	a1	a2
Constant	A0	0.0	0.0
Linear	A0	A1-A0	A2-A0
Wrap Shortest	A0	(A1-A0)+1 (A1-A0) <= -0.5 (A1-A0)-1 (A1-A0) >= 0.5 (A1-A0) otherwise	(A2-A0)+1 (A2-A0) <= -0.5 (A2-A0)-1 (A2-A0) >= 0.5 (A2-A0) otherwise

Point Sprites

Normally all vertex attributes (including texture coordinates) other than position are simply replicated from the incoming point center vertex to the generated point object (corner) vertices. However, both DX9 and OGL support “sprite points”, where some/all texture coordinates are replaced with full-scale 2D texture coordinates.

A 32-bit **PointSprite TextureCoordinate Enable** bit mask controls whether the corresponding vertex attribute is to be replaced by a sprite point texture coordinate. The global (not per-attribute) **Point Sprite TextureCoordinate Origin** field controls how the point object vertex (top/bottom, left/right) texture coordinates are generated:

UPPERLEFT	Left	Right
Top	(0,0,0,1)	(1,0,0,1)
Bottom	(0,1,0,1)	(1,1,0,1)

LOWERLEFT	Left	Right
Top	(0,1,0,1)	(1,1,0,1)
Bottom	(0,0,0,1)	(1,0,0,1)

The state used by “setup backend” is defined by the following inline state packet.

3DSTATE_SBE

The state used by “setup backend” is defined by the following inline state packet.

3DSTATE_SBE_SWIZ

SBE Unit also receives 3DSTATE_INT which is transparent to SW. 3DSTATE_INT provides 3DSTATE_VS, 3DSTATE_DS, and 3DSTATE_GS fields.

Signal [CHV, BSW]	SBE_INT::Vertex URB Entry Read Length
Description	Specifies the amount of URB data read for each Vertex URB entry, in 256-bit register increments.
Formula	= (3DSTATE_SBE::Force Vertex URB Entry Read Length == Force) ? 3DSTATE_SBE::Vertex URB Entry Read Length : 3DSTATE_GS::GS_Enable ? 3DSTATE_GS::Vertex URB Entry Output Length : 3DSTATE_DS::DS_Enable ? 3DSTATE_DS::Vertex URB Entry Output Length : 3DSTATE_VS::Vertex URB Entry Output Length

Signal [CHV, BSW]	SBE_INT::Vertex URB Entry Read Offset
Description	Specifies the offset (in 256-bit units) at which Vertex URB data is to be read from the URB
Formula	= (3DSTATE_SBE::Force Vertex URB entry Offset == Force) ? 3DSTATE_SBE::Vertex URB Entry Read Offset: 3DSTATE_GS::GS_Enable ? 3DSTATE_GS:: Vertex URB Entry Output Read Offset: 3DSTATE_DS:DS_Enable ? 3DSTATE_DS:: Vertex URB Entry Output Read Offset : 3DSTATE_VS:: Vertex URB Entry Output Read Offset

Signal [CHV, BSW]	SBE_INT::PrimId_override
Description	When true indicates that SBE provides the Primitive ID.
Formula	= 3DSTATE_GS::GS_Enable ? false : 3DSTATE_SBE::Primitive ID Override Component Select !=0

Barycentric Attribute Interpolation

Given hardware clipper and setup, some of the previous flexibility in the algorithm used to interpolate attributes is no longer available. Hardware uses barycentric parameters to aid in attribute interpolation, and these parameters are computed in hardware per-pixel (or per-sample) and delivered in the thread payload to the pixel shader. Also delivered in the payload are a set of vertex deltas (a0, a1, and a2) per channel of each attribute.

There are six different barycentric parameters that can be enabled for delivery in the pixel shader payload. These are enabled via the **Barycentric Interpolation Mode** bits in 3DSTATE_WM.

In the pixel shader kernel, the following computation is done for each attribute channel of each pixel/sample given the corresponding attribute channel a0/a1/a2 and the pixel/sample's b1/b2 barycentric parameters, where A is the value of the attribute channel at that pixel/sample:

A = a0 + (a1 * b1) + (a2 * b2)

Depth Offset

The state for depth offset in 3DSTATE_SF controls the depth offset function. Since this function was previously contained in the Windower stage, refer to the "Depth Offset" section in the Windower chapter for more details on this function.

Other SF Functions

The only other SF-related function is statistics gathering.

Statistics Gathering

The SF stage itself does not have any associated pipeline statistics; however, it counts the number of objects being output by the clipper on the clipper's behalf, since it is less feasible to have the CLIP unit figure out how many objects have been output by a clip thread. It is easy for the SF unit to count the number of objects it receives from the CLIP stage since it is decomposing the output primitive topologies into objects anyway.

If the **Statistics Enable** bit is set in SF_STATE, then SF will increment the CL_PRIMITIVES_COUNT Register (see Memory Interface Registers in Volume Ia, *GPU*) once for each object in each primitive topology it receives from the CLIP stage. This bit should always be set if clipping is enabled and pipeline statistics are desired.

Software should always clear the **Statistics Enable** bit in SF_STATE if the clipper is disabled since objects SF receives are not considered "primitives output by the clipper" unless the clipper is enabled. Note that the clipper can be disabled either using bypass mode via a PIPELINE_STATE_POINTERS command with **Clip Enable** clear *or* by setting **Clip Mode** in CLIP_STATE to CLIPMODE_ACCEPT_ALL.

Windower (WM) Stage

This topic is currently under development.

Overview

As mentioned in the *SF Unit* chapter, the SF stage prepares an object for scan conversion by the Window/Masker (WM) unit Refer to the *SF Unit* chapter for details on the screen-space geometry of objects to be rendered The WM unit uses the parameters provided by the SF unit in the object-specific rasterization algorithms.

The WM stage of the 3D pipeline performs the following operations (at a high level)

- Pre-scan-conversion modification of some primitive attributes, including
    - Application of Depth Offset to the position Z attribute
- Scan-conversion of the various primitive types, including
    - 2D clipping to the scissor/draw rectangle intersection
- Spawning of Pixel Shader (PS) threads to process the pixels resulting from scan-conversion

The spawned Pixel Shader (PS) threads are responsible for the following (high-level) operations

- interpolation of vertex attributes (other than X,Y,Z) to the pixel location
- performing any “Pixel Shader” operations dictated by the API PS program
    - Using the Sampler shared function to sample data from “texture” surfaces
    - Using the DataPort to perform general memory I/O
- Submitting the shaded pixel results to the DataPort for any subsequent “blending” (aka Output Merger) operation and write to the RenderCache.

The WM unit keeps a scoreboard of pixels being processed in outstanding PS threads in order to guarantee in-order rasterization results This allows the WM unit to overlap processing of several objects.

Inputs from SF to WM

The outputs from the SF stage to the WM stage are mostly comprised of implementation-specific information required for the rasterization of objects The types of information is summarized below, but as the interface is not exposed to software a detailed discussion is not relevant to this specification.

- PrimType of the object
- VPIndex, RTAIndex associated with the object
- Handle of the Primitive URB Entry (PUE) that was written by the SF (Setup) thread This handle will be passed to all WM (PS) threads spawned from the WM's rasterization process.
- Information regarding the X,Y extent of the object (e.g., bounding box, etc.)
- Edge or line interpolation information (e.g., edge equation coefficients, etc.)
- Information on where the WM is to start rasterization of the object
- Object orientation (front/back-facing)
- Last Pixel indication (for line drawing)

Windower Pipelined State

3DSTATE_WM

The following inline state packets define the state used by the windower stage for different generations.

3DSTATE_WM

Programming Note	
Context:	XYZ
Note: WM Unit also receives 3DSTATE_WM_HZ_OP, 3DSTATE_RASTER, 3DSTATE_MULTISAMPLE, 3DSTATE_WM_CHROMAKEY, 3DSTATE_PS_BLEND, and 3DSTATE_PS_EXTRA	

Signal [CHV, BSW]	WM_INT::ThreadDispatchEnable
Description	This bit, if set, indicates that it is possible for a PS thread to modify a render target.
Formula	= (3DSTATE_WM::ForceThreadDispatch == ON) || ((3DSTATE_WM::ForceThreadDispatch != OFF) && ! WM_INT::WM_HZ_OP && 3DSTATE_PS_EXTRA::PixelShaderValid && ((!3DSTATE_PS_EXTRA::PixelShaderDoesNotWriteRT && 3DSTATE_PS_BLEND::HasWriteableRT) ||

	(3DSTATE_PS_EXTRA::PixelShaderHasUAV) || WM_INT:: Pixel Shader Kill Pixel || (WM_INT::Pixel Shader Computed Depth Mode != PSCDEPTH_OFF && (WM_INT::Depth Test Enable || WM_INT::Depth Write Enable)) || (3DSTATE_PS_EXTRA::Computed Stencil && WM_INT::Stencil Test Enable) || (3DSTATE_WM::EDSC_Mode == 1 && (WM_INT::Depth Test Enable || WM_INT::Depth Write Enable || WM_INT::Stencil Test Enable)) || (WM_INT::RT Independent Rasterization Enable)))

Signal [CHV, BSW]	WM_INT::Pixel Shader Computed Depth Mode
Description	This field specifies the computed depth mode for the pixel shader.
Formula	= (3DSTATE_PS_EXTRA::ForceComputedDepth == Force) ? 3DSTATE_PS_EXTRA::Pixel Shader Computed Depth Mode : (WM_INT::WM_HZ_OP || WM_INT::RT Independent Rasterization Enable) ? PSCDEPTH_OFF: 3DSTATE_PS_EXTRA::Pixel Shader Computed Depth Mode

Signal [CHV, BSW]	WM_INT::Pixel Shader Uses Source Depth
Description	This bit, if ENABLED, indicates that the PS kernel requires the source depth value (vPos.z) to be passed in the payload.
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Source Depth

Signal [CHV, BSW]	WM_INT::Pixel Shader Uses Source W

Description	This bit, if ENABLED, indicates that the PS kernel requires the interpolated source W value (vPos.w) to be passed in the payload
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Source W

Signal [CHV, BSW]	WM_INT::Pixel Shader Uses Input Coverage Mask
Description	This bit, if ENABLED, indicates that the PS kernel requires the input coverage mask to be passed in the payload.
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Uses Input Coverage Mask

Signal [CHV, BSW]	WM_INT::Multisample Rasterization Mode
Description	This field determines whether multisample rasterization is enabled and how pixel sample points are defined.
Formula	See Table below: WM_INT::Multisample Rasterization Mode

WM_INT::Multisample Rasterization Mode

3DSTATE_RASTER:: Force Multisampling	Force	Force	Force	Force	Normal	Normal	Normal
3DSTATE_RASTER:: DX Multisample Rasterization Mode	MSRASTMODE_OFF_PIXEL	MSRASTMODE_OFF_PATTERN	MSRASTMODE_ON_PIXEL	MSRASTMODE_ON_PATTERN	*	*	*
WM_INT::WM_HZ_OP	*	*	*	*	True	True	False
3DSTATE_WM_HZ_OP:: Number of Multisamples	*	*	*	*	> NUMSAMPLES_1	NUMSAMPLES_1	*
WM_INT::Multisample Rasterization Mode	OFF_PIXEL	OFF_PATTERN	ON_PIXEL	ON_PATTERN	ON_PATTERN	ON_PIXEL	Determined from Table 1 in 3D Pipeline Windower

							[CHV, BSW] > Multisampling [CHV, BSW] > Multisample Modes/State)

Note: OFF_PIXEL, OFF_PATTERN, ON_PIXEL, ON_PATTERN modes are described in 3D Pipeline Windower [CHV, BSW] > Multisampling [CHV, BSW] > Multisample Modes/State.

Signal [CHV, BSW]	WM_INT::Multisample Dispatch Mode
Description	This bit, determines how PS threads are dispatched
Formula	= (WM_INT::RT Independent Rasterization Enable)? PerPixel: (3DSTATE_PS_EXTRA::PixelShaderIsPerSample) ? PerSample : PerPixel

Signal [CHV, BSW]	WM_INT::Pixel Shader Kill Pixel
Description	This bit, if ENABLED, indicates that the PS kernel or color calculator has the ability to kill (discard) pixels or samples, *other than due to depth or stencil testing*.
Formula	= (3DSTATE_WM::ForceKillPixel == ON) || ((3DSTATE_WM::ForceKillPixel != Off) && ! WM_INT::WM_HZ_OP && ! 3DSTATE_WM::EDSC_Mode == 2 && (WM_INT::Depth Write Enable || WM_INT::Stencil Write Enable) && (3DSTATE_PS_EXTRA::PixelShaderKillsPixels || 3DSTATE_PS_EXTRA:: **oMask Present to RenderTarget** || 3DSTATE_PS_BLEND::AlphaToCoverageEnable || 3DSTATE_PS_BLEND::AlphaTestEnable ||

	3DSTATE_WM_CHROMAKEY::ChromaKeyKillEnable))

Signal [CHV, BSW]	WM_INT::Early Depth/Stencil Control
Description	This field specifies the behavior of early depth/stencil test.
Formula	= (WM_INT::WM_HZ_OP) ? EDSC_NORMAL : WM_INT::RT Independent Rasterization Enable ? EDSC_PSEXEC : 3DSTATE_WM::Early Depth/Stencil Control

Signal [CHV, BSW]	WM_INT::RT Independent Rasterization Enable
Description	Enables Render Target Independent Rasterization.
Formula	= (WM_INT::WM_HZ_OP ? Disable : (3DSTATE_RASTER::ForceSampleCount != NUMRASTSAMPLES_0) ? Enable : Disable

Signal [CHV, BSW]	WM_INT::Statistics Enable
Description	Enables Statistics
Formula	= (WM_INT::WM_HZ_OP) ? Disable : 3DSTATE_WM:: Statistics Enable

Signal [CHV, BSW]	WM_INT::Polygon Stipple Enable
Description	Enables Poly Stipple
Formula	= (WM_INT::WM_HZ_OP) ? Disable : 3DSTATE_WM::Polygon Stipple Enable

Signal [CHV, BSW]	WM_INT::WM_HZ_OP
Description	Enables WM_HZ_OP
Formula	= (3DSTATE_WM_HZ_OP::DepthBufferClear || 3DSTATE_WM_HZ_OP::DepthBufferResolve || 3DSTATE_WM_HZ_OP::Hierarchical Depth Buffer Resolve Enable || 3DSTATE_WM_HZ_OP::StencilBufferClear) ? Enable : Disable

Signal	WM_INT:: Pixel Location
Description	Sets the input pixel location to Center if UL and doing multisampling
Formula	(3DSTATE_MULTISAMPLE::Pixel Location && 3DSTATE_MULTISAMPLE::Pixel Position Offset Enable && WM_MULTISAMPLE_INT::Number of Multisamples > 0) ? 0 : 3DSTATE_MULTISAMPLE::Pixel Location

3DSTATE_SAMPLE_MASK

The following inline state packets define the sample mask state used by the windower stage for different generations.

3DSTATE_SAMPLE_MASK

Signal [CHV, BSW]	WM_INT:: Sample Mask Enable
Description	Sets Sample Mask used in rasterization

Formula	Switch(WM_MULTISAMPLE_INT::Number of Multisamples
	{
	Case NUMSAMPLES_1: WM_INT:: Sample Mask Enable = 0x0001; break;
	Case NUMSAMPLES_2: WM_INT:: Sample Mask Enable = 0x0003; break;
	Case NUMSAMPLES_4: WM_INT:: Sample Mask Enable = 0x000F; break;
	Case NUMSAMPLES_8: WM_INT:: Sample Mask Enable = 0x00FF; break;
	}

Signal [CHV, BSW]	WM_INT:: Sample Mask
Description	Sets Sample Mask used in rasterization
Formula	= WM_INT:: Sample Mask Enable & (WM_INT::WM_HZ_OP) ? 3DSTATE_WM_HZ_OP::Sample Mask: 3DSTATE_SAMPLE_MASK::Sample Mask)

3DSTATE_WM_CHROMAKEY

3DSTATE_WM_HZ_OP

State Restrictions

State	Restriction
3DSTATE_PS::Render Target Fast Clear Enable	Must be disabled
3DSTATE_PS:: Render Target Resolve Enable	Must be disabled
3DSTATE_WM:: Legacy Depth Buffer Clear	Must be disabled
3DSTATE_WM:: Legacy Depth Buffer Resolve	Must be disabled
3DSTATE_WM:: Legacy Hierarchical Depth Buffer Resolve Enable	Must be disabled
3DSTATE_MULTISAMPLE::Pixel Location	Must be set according to the API being used.
3DSTATE_CLEAR_PARAMS CHV, BSW:Depth Clear Value	Must be programmed according to the API when 3DSTATE_WM_HZ_OP::Depth Buffer Clear is set
3DSTATE_CLEAR_PARAMS CHV, BSW:Depth Clear Value Valid	Must be enabled when 3DSTATE_WM_HZ_OP::Depth Buffer Clear is set

State Overrides

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
SF_INT:: Statistics Enable	Disable	Disable	Disable	Disable
SF_INT:: View Transform Enable	Disable	Disable	Disable	Disable
SF_INT::Multisample Rasterization Mode	(3DSTATE_WM_HZ_OP::NumberOfSamples > 1) ? ON_PATTERN : ON_PIXEL	(3DSTATE_WM_HZ_OP::NumberOfSamples > 1) ? ON_PATTERN : ON_PIXEL	(3DSTATE_WM_HZ_OP::NumberOfSamples > 1) ? ON_PATTERN : ON_PIXEL	(3DSTATE_WM_HZ_OP::NumberOfSamples > 1) ? ON_PATTERN : ON_PIXEL
SF_INT::Cull Mode	CULLMODE_BACK	CULLMODE_BACK	CULLMODE_BACK	CULLMODE_BACK
SF_INT::Scissor Rectangle Enable	3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable	3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable	3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable	3DSTATE_WM_HZ_OP:: Scissor Rectangle Enable
SF_INT::RT Independent Rasterization Enable	Disable	Disable	Disable	Disable
SF_INT::FrontFace	SOLID	SOLID	SOLID	SOLID

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
Fill Mode				
SF_INT::FrontWinding	FRONTWINDING_CW	FRONTWINDING_CW	FRONTWINDING_CW	FRONTWINDING_CW
SF_INT::Render Target Array index	0	0	0	0
SF_INT::Viewport index	0	0	0	0
SF_INT:: Geometry Hashing Disable	Disable	Disable	Disable	Disable
WM_INT::StencilTestEnable	Enable	Stencil buffer Clear ? Enable : Disable	Disable	Disable
WM_INT::StencilWriteEnable	Enable	Stencil buffer Clear ? Enable : Disable	Disable	Disable
WM_INT::DepthTestEnable	Disable	Disable	Enable	Disable
WM_INT::DepthWriteEnable	Depth buffer Clear ? Enable : Disable	Enable	Enable	Enable
WM_INT::DepthTestFunction	NEVER	NEVER	NEVER	NEVER
WM_INT::StencilTestFunction	ALWAYS	Stencil buffer Clear ? ALWAYS: No Override	No Override	No Override
WM_INT::StencilPassDepthPassOp	REPLACE	Stencil buffer Clear ? REPLACE: No Override	No Override	No Override
WM_INT:: Statistics Enable	Disable	Disable	Disable	Disable

State	Stencil buffer Clear	Depth buffer clear	Depth Buffer Resolve Enable	Hierarchical Depth Buffer Resolve Enable
WM_INT::ThreadDispatchEnable	Disable	Disable	Disable	Disable
WM_INT:: Pixel Shader Kill Pixel	Disable	Disable	Disable	Disable
WM_INT:: Pixel Shader Computed Depth Mode	PSCDEPTH_OFF	PSCDEPTH_OFF	PSCDEPTH_OFF	PSCDEPTH_OFF
WM_INT::RT Independent Rasterization Enable	Disable	Disable	Disable	Disable
WM_INT::Polygon Stipple Enable	Disable	Disable	Disable	Disable
WM_INT::Multisample Rasterization Mode	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)	NumberOfSamples > 0 ? ON_PATTERN : ON_PIXEL)
MULTISAMPLE_INT::Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples	3DSTATE_WM_HZ_OP:: Number of Multisamples
WM_INT::Sample Mask	3DSTATE_WM_HZ_OP::Sample Mask	3DSTATE_WM_HZ_OP::Sample Mask	3DSTATE_WM_HZ_OP::Sample Mask	3DSTATE_WM_HZ_OP::Sample Mask
WM_INT::Early Depth/Stencil Control	EDSC_NORMAL	EDSC_NORMAL	EDSC_NORMAL	EDSC_NORMAL
WM_INT:: Full Surface Depth Clear	Depth buffer clear ? WM_HZ_OP:: Full Surface Depth Clear : Disable	WM_HZ_OP:: Full Surface Depth Clear	Disable	Disable
WM_INT:: Full Surface Depth Clear	Depth buffer clear ? WM_HZ_OP:: Full Surface Depth Clear : Disable	WM_HZ_OP:: Full Surface Depth Clear	Disable	Disable

3DSTATE_WM_DEPTH_STENCIL

Signal [CHV, BSW]	WM_INT::StencilWriteEnable
Description	Enables writes to the Stencil Buffer
Formula	= 3DSTATE_STENCIL_BUFFER::STENCIL_BUFFER_ENABLE && 3DSTATE_DEPTH_BUFFER::STENCIL_WRITE_ENABLE && ((WM_INT::WM_HZ_OP ? Use the WM_INT::StencilWriteEnable from WM_HZ_OP table : WM_INT::StencilTestEnable && 3DSTATE_WM_DEPTH_STENCIL::StencilBufferWriteEnable))

Signal [CHV, BSW]	WM_INT::StencilTestEnable
Description	Enables Stencil Test
Formula	= 3DSTATE_STENCIL_BUFFER::STENCIL_BUFFER_ENABLE && (WM_INT::WM_HZ_OP ? Use the WM_INT::StencilTestEnable from WM_HZ_OP table : (3DSTATE_WM_DEPTH_STENCIL::StencilTestEnable && !WM_INT::RT Independent Rasterization Enable))

Signal [CHV, BSW]	WM_INT::DepthTestEnable
Description	Enables Depth Test
Formula	= (3DSTATE_DEPTH_BUFFER::SURFACE_TYPE != NULL) && (WM_INT::WM_HZ_OP ? Use the WM_INT::DepthTestEnable from WM_HZ_OP table : (3DSTATE_WM_DEPTH_STENCIL::DepthTestEnable && ! WM_INT::RT Independent Rasterization Enable))

Signal [CHV, BSW]	WM_INT::DepthWriteEnable
Description	Enables Depth Write
Formula	= (3DSTATE_DEPTH_BUFFER::SURFACE_TYPE != NULL) && 3DSTATE_DEPTH_BUFFER::DEPTH_WRITE_ENABLE && (WM_INT::WM_HZ_OP ? Use the WM_INT::DepthWriteEnable from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::DepthWriteEnable)

Signal [CHV, BSW]	WM_INT::DepthTestFunction
Description	Depth Test Function
Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::DepthTestFunction from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::DepthTestFunction

Signal [CHV, BSW]	WM_INT::StencilTestFunction
Description	Stencil Test Function
Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::StencilTestFunction from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::StencilTestFunction

Signal [CHV, BSW]	WM_INT::StencilPassDepthPassOp
Description	StencilPassDepthPassOp
Formula	WM_INT::WM_HZ_OP ? Use the WM_INT::StencilPassDepthPassOp from WM_HZ_OP table : 3DSTATE_WM_DEPTH_STENCIL::StencilPassDepthPassOp

Signal [CHV, BSW]	WM_INT::Stencil Test Mask
Description	Stencil test Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Stencil Test Mask

Signal [CHV, BSW]	WM_INT::Stencil Write Mask
Description	Stencil Write Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Stencil Write Mask

Signal [CHV, BSW]	WM_INT::BackFace Stencil Test Mask
Description	Stencil test Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL:: Backface Stencil Test Mask

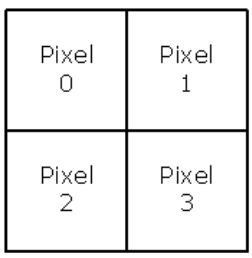
Signal [CHV, BSW]	WM_INT:: BackFace Stencil Write Mask
Description	Stencil Write Mask
Formula	= 3DSTATE_WM_HZ_OP::StencilClear ? 0xFF : 3DSTATE_WM_DEPTH_STENCIL::Backface Stencil Write Mask

Rasterization

The WM unit uses the setup computations performed by the SF unit to rasterize objects into the corresponding set of pixels Most of the controls regarding the screen-space geometry of rendered objects are programmed via the SF unit.

The rasterization process generates pixels in 2x2 groups of pixels called *subspans* (see *Pixels with a SubSpan* below) which, after being subjected to various inclusion/discard tests, are grouped and passed to spawned Pixel Shader (PS) threads for subsequent processing Once these PS threads are spawned, the WM unit provides only bookkeeping functions on the pixels Note that the WM unit can proceed on to rasterize subsequent objects while PS threads from previous objects are still executing.

Pixels with a SubSpan



Drawing Rectangle Clipping

The Drawing Rectangle defines the maximum extent of pixels which can be rendered Portions of objects falling outside the Drawing Rectangle will be clipped (pixels discarded) Implementations will typically discard objects falling completely outside of the Drawing Rectangle as early in the pipeline as possible There is no control to turn off Drawing Rectangle clipping – it is unconditional.

For the purposes of clipping, the Drawing Rectangle must itself be clipped to the destination buffer extents (The Drawing Rectangle Origin, used to offset relative X,Y coordinates earlier in the pipeline, is permitted to lie offscreen). The **Clipped Drawing Rectangle X,Y Min,Max** state variables (programmed via 3DSTATE_DRAWING_RECTANGLE – See *SF Unit*) defines the intersection of the Drawing Rectangle and the Color Buffer It is specified with non-negative integer pixel coordinates relative to the Destination Buffer upper-left origin.

Pixels with coordinates outside of the Drawing Rectangle cannot be rendered (i.e., the rectangle is inclusive) For example, to render to a full-screen 1280x1024 buffer, the following values would be required: Xmin=0, Ymin=0, Xmax=1279 and Ymax=1023

For “full screen” rendering, the Drawing Rectangle coincides with the screen-sized buffer For “front-buffer windowed” rendering it coincides with the destination “window”.

Line Rasterization

See *SF Unit* chapter for details on the screen-space geometry of the various line types.

Coverage Values for Anti-Aliased Lines

The WM unit is provided with both the **Line Anti-Aliasing Region Width** and **Line End Cap Anti-aliasing Region Width** state variables (in WM_STATE) in order to compute the coverage values for anti-aliased lines.

3DSTATE_AA_LINE_PARAMS [CHV, BSW]

3DSTATE_AA_LINE_PARAMETERS

The slope and bias values should be computed to closely match the reference rasterizer results Based on empirical data, the following recommendations are offered:

The final alpha for the center of the line needs to be 148 to match the reference rasterizer In this case, the Lo to edge 0 and edge 3 will be the same Since the alpha for each edge is multiplied together, we get:

edge0alpha * edge1alpha = 148/255 = 0.580392157

Since edge0alpha = edge3alpha we get:

$(edge0alpha)^2 = 0.580392157$

edge0alpha = sqrt(0.580392157) = 0.761834731 at the center pixel

The desired alpha for pixel 1 = 54/255 = 0.211764706

The slope is (0.761834731 – 0.211764706) = 0.550070025

Since we are using 8 bit precision, the slope becomes

AA Coverage [EndCap] Slope = 0.55078125

The alpha value for Lo = 0 (second pixel from center) determines the bias term and is equal to

(0.211764706 – 0.550070025) = -0.338305319

With 8 bits of precision the programmed bias value

Line Stipple

Line stipple, controlled via the **Line Stipple Enable** state variable in WM_STATE, discards certain pixels that are produced by non-AA line rasterization.

The line stipple rule is specified via the following state variables programmed via 3DSTATE_LINE_STIPPLE: the 16-bit **Line Stipple Pattern** (p), **Line Stipple Repeat Count** l, and **Line Stipple Inverse Repeat Count.** Sofware must compute **Line Stipple Inverse Repeat Count** as 1.0f / **Line Stipple Repeat Count** and then converted from float to the required fixed point encoding (see 3STATE_LINE_STIPPLE).

The WM unit maintains an internal Line Stipple Counter state variable (s) The initial value of s is zero; s is incremented after production of each pixel of a line segment (pixels are produced in order, beginning at the starting point and working towards the ending point). S is reset to 0 whenever a new primitive is processed (unless the primitive type is LINESTRIP_CONT or LINESTRIP_CONT_BF), and before every line segment in a group of independent segments (LINELIST primitive).

During the rasterization of lines, the WM unit computes:

$$b = \lfloor s/r \rfloor \bmod 16,$$

A pixel is rendered if the bth bit of p is 1, otherwise it is discarded. The bits of p are numbered with 0 being the least significant and 15 being the most significant.

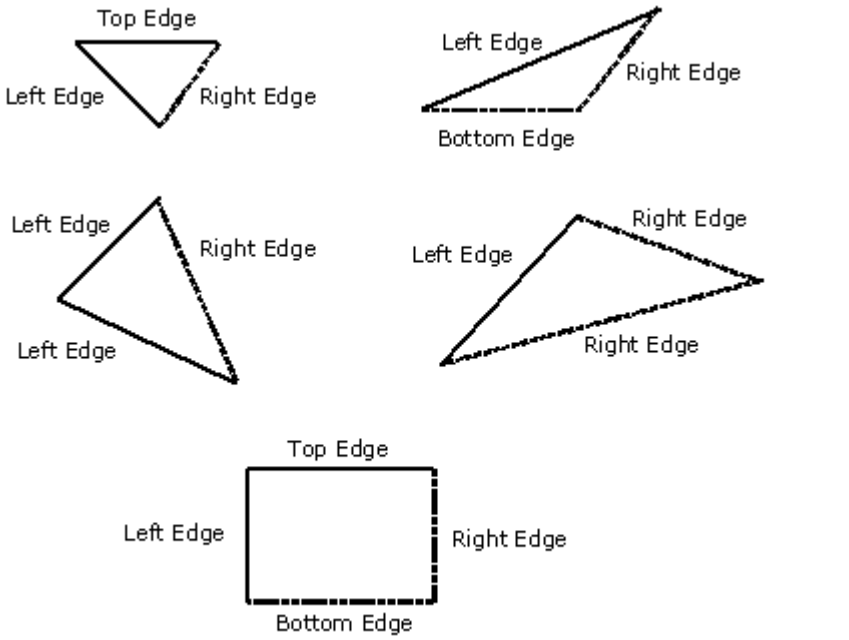
3DSTATE_LINE_STIPPLE

Polygon (Triangle and Rectangle) Rasterization

The rasterization of LINE, TRIANGLE, and RECTANGLE objects into pixels requires a “pixel sampling grid” to be defined This grid is defined as an axis-aligned array of pixel sample points spaced exactly 1 pixel unit apart If a sample point falls within one of these objects, the pixel associated with the sample point is considered “inside” the object, and information for that pixel is generated and passed down the pipeline

For TRIANGLE and RECTANGLE objects, if a sample point intersects an edge of the object, the associated pixel is considered “inside” the object if the intersecting edge is a “left” or “top” edge (or, more exactly, the intersected edge is not a “right” or “bottom” edge) Note that “top” and “bottom” edges are by definition exactly horizontal. See *TRIANGLE and RECTANGLE Edge Types* below for the edge types for representative TRIANGLE and RECTANGLE objects (solid edges are inclusive, dashed edges are exclusive).

TRIANGLE and RECTANGLE Edge Types



Polygon Stipple

The *Polygon Stipple* function, controlled via the **Polygon Stipple Enable** state variable in WM_STATE, allows only selected pixels of a repeated 32x32 pixel pattern to be rendered Polygon stipple is applied only to the following primitive types:

3DPRIM_POLYGON
3DPRIM_TRIFAN
3DPRIM_TRILIST
3DPRIM_TRISTRIP
3DPRIM_TRISTRIP_REVERSE

Note that the 3DPRIM_TRIFAN_NOSTIPPLE object is never subject to polygon stipple.

The stipple pattern is defined as a 32x32 bit pixel mask via the 3DSTATE_POLY_STIPPLE_PATTERN command. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

The origin of the pattern is specified via **Polygon Stipple X,Y Offset** state variables programmed via the 3DSTATE_POLY_STIPPLE_OFFSET command The offsets are pixel offsets from the Color Buffer origin to the upper left corner of the stipple pattern. This is a non-pipelined command which incurs an implicit pipeline flush when executed.

3DSTATE_POLY_STIPPLE_OFFSET

3DSTATE_POLY_STIPPLE_PATTERN

Multisampling

The multisampling function has two components:

- **Multisample Rasterization**: multisample rasterization occurs at a subpixel level, wherein each pixel consists of a number of "samples" at state-defined positions within the pixel footprint. Coverage of the primitive as well as color calculator operations (stencil test, depth test, color buffer blending, etc.) are done at the sample level. In addition the pixel shader itself can optionally run at the sample level depending on a separate state field.
- **Multisample Render Targets (MSRT)**: The render targets, as well as the depth and stencil buffers, now have the ability to store per-sample values. When combined with multisample rasterization, color calculator operations such as stencil test, depth test, and color buffer blending are done with the destination surface containing potentially different values per sample.

3DSTATE_MULTISAMPLE

Signal [CHV, BSW]	WM_MULTISAMPLE_INT::Number of Multisamples
Description	Set the number of multisamples
Formula	= (WM_INT::WM_HZ_OP) ? 3DSTATE_ WM_HZ_OP::Number of Multisamples : 3DSTATE_MULTISAMPLE::Number of Multisamples

3DSTATE_SAMPLE_PATTERN

Multisample ModesState

A number of state variables control the operation of the multisampling function. The following table indicates the states and their location. Refer to the state definition for more details.

State Element	Source	Description
	WM_INT::Multisample Rasterization Mode	Controls whether rasterization of non-lines is performed on a pixel or sample basis (PIXEL vs. PATTERN), and whether multisample rasterization of lines is enabled (OFF vs. ON). From this generation forward, this state element becomes an internal signal computed by other state variables (also listed here) unless certain modes are set, which can be seen in the WM_INT equation for the signal.
	WM_INT::Multisample Dispatch Mode	
Number of Multisamples	3DSTATE_MULTISAMPLE and SURFACE_STATE	Indicates the number of samples per pixel contained on the surface. This field in 3DSTATE_MULTISAMPLE must match the corresponding field in SURFACE_STATE for each render target. The depth, hierarchical depth, and stencil buffers inherit this field from 3DSTATE_MULTISAMPLE.

State Element	Source	Description
	3DSTATE_RASTER::ForcedSampleCount	
	3DSTATE_RASTER::ForcedSampleCount != NUMRASTSAMPLES_0	
Pixel Location	3DSTATE_MULTISAMPLE	Indicates the subpixel location where values specified as "pixel" are sampled. This is either the upper left corner or the center.
	3DSTATE_SAMPLE_PATTERN	
	3DSTATE_SAMPLE_PATTERN	
API Mode	3DSTATE_RASTER	One of the deciding factors of what the Multisample Rasterization Mode should be according to WM_INT::Multisample Rasterization Mode. Software sets this field according to the API's version.
DX Multisample Rasterization Enable	3DSTATE_RASTER	Controls ON/OFF part of Multisample Rasterization Mode, depending on the API Mode according to WM_INT::Multisample Rasterization Mode.

This table does not apply if (3DSTATE_RASTER::ForceMultisampleRasterMode == Force) or (WM_INT::WM_HZ_OP == true).

Table 1: Multisample Rasterization Modes [CHV, BSW]

Number of Multisamples	NUMSAMPLES_1	NUMSAMPLES_1	> NUMSAMPLES_1	> NUMSAMPLES_1	Any	Any	Any	Any
DX Multisample Rasterization Enable	0	1	0	1	0	1	0	1
Rast Number of Samples	Disabled	Disabled	Disabled	Disabled	NUMRAST SAMPLES_1	NUMRAST SAMPLES_1	> NUMRAST SAMPLES_1	> NUMRAST SAMPLES_1
API Mode == DX9.0/OGL	OFF_PIXEL	OFF_PIXEL	OFF_PIXEL	ON_PATTERN	Invalid	Invalid	Invalid	Invalid
API Mode	OFF_PIXEL	ON_PIXEL	OFF_PIXEL	ON_PATTER	OFF_PIX	Invalid	Invalid	Invalid

== DX10.0				N	EL			
API Mode == DX10.1+	OFF_PIXEL	ON_PIXEL	OFF_PATTERN	ON_PATTERN	OFF_PIXEL	ON_PIXEL	OFF_PATTERN	ON_PATTERN

Definitions for lines terms used in Table 2 through Table 4:

- **Legacy Lines:** Way of drawing lines that allows Diamond Lines (SF_STATE::Line Width == 0.0), Non-anti-aliased Wide Lines (SF_STATE::Line Width != 0.0), and Line Stippling (3DSTATE_WM:: Line Stipple Enable == 1).
- **AA Lines:** Way of drawing lines that allows Anti-aliased line. These are lines rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices with coverage value (referred to as Anti-alias Alpha) computed per pixel.

Project	AA Line Support Requirement
CHV, BSW	SF_INT::Anti-aliasing Enable == 1

- **MSAA Lines:** Way of drawing lines that allows Multisample Anti-aliased lines. These are lines rendered as rectangles that are centered on, and aligned to, the line joining the endpoint vertices, but no Anti alias alpha coverage is computed.

Table 2: Type of Line Algorithm Given an Arrangement of State Variables [CHV, BSW]

Multisample Rasterization Mode	Anti-Aliasing Enable	SF_STATE::Line Width	Line Algorithm
OFF_*	0	Non-Zero	Non-Anti-aliased Wide Lines
OFF_*	0	0.0	Diamond Lines
OFF_*	1	Non-Zero	See Note A below.
OFF_*	1	0.0	Diamond Lines
ON_*	*	*	MSAA Lines

Note A: Anti-Aliasing Details for Table 2

Anti-Aliasing Details
Anti-Aliased Lines with Alpha Coverage

Table 3: Multisample Modes with RTIR Disabled [CHV, BSW]

Number of Multisamples	MS RAST MODE	MS DISP MODE	HW Mode
NUMSAMPLES_1	OFF_PIXEL	PERSAMPLE	**Legacy Non-MSAA Mode** 1X rasterization, using Pixel Location

			Legacy lines or AA-line rasterization 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PIXEL	PERSAMPLE	**1X Multisampling Mode** 1X rasterization, using Pixel Location MSAA lines only, using Pixel Location 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	-	PERPIXEL	Treated the same as PERSAMPLE
	ON_PATTERN	-	Invalid
	OFF_PATTERN	-	Invalid
n where n > NUMSAMPLES_1	OFF_PIXEL	PERPIXEL	**MSRT Only, PerPixel PS** 1X rasterization, using Pixel Location See Note B below. 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**MSRT Only, PerSample PS** 1X rasterization, using Pixel Location See Note B below. nX PS, all samples at Pixel Location nX output merge, eval Depth at Pixel Location
	ON_PIXEL	PERPIXEL	**Multibuffering MSAA, PerPixel PS** 1X rasterization, using Pixel Location MSAA lines only 1X PS, sample at Pixel Location 4X output merge, eval Depth at Pixel Location
		PERSAMPLE	**Multibuffering MSAA, PerSample PS** 1X rasterization, using Pixel Location MSAA lines only nX PS, all samples at Pixel Location

			nX output merge, eval Depth at Pixel Location
	OFF_PATTERN	PERPIXEL	**Mixed Mode, PerPixel PS** See Note B below. Non-Lines: nX rasterization, using Sample Offsets 1X PS, sample at Pixel Location nX output merge, eval depth at Sample Offsets
		PERSAMPLE	**Mixed Mode, PerSample PS** See Note B below. Non-Lines: nX rasterization, using Sample Offsets nX PS, sample at Pixel Location or Sample Offsets nX output merge, eval depth at Sample Offsets
	ON_PATTERN	PERPIXEL	**Pattern MSAA, PerPixel PS** nX rasterization, using Sample Offsets MSAA lines only 1X PS, sample at Pixel Location nX output merge, eval depth at Sample Offsets
		PERSAMPLE	**Pattern MSAA, PerSample PS** nX rasterization, using Sample Offsets MSAA lines only nX PS, sample at Pixel Location or Sample Offsets nX output merge, eval depth at Sample Offsets

Note B: Line Details for Table 3 and Table 4

Line Details
Legacy lines or AA-line rasterization. For PERPIXEL or PERSAMPLE in Table 3 use pixel location. For OFF_PATTERN in Table 4 use pixel location.

Table 4: Multisample Modes with RTIR Enabled [CHV, BSW]

Rast Number of Samples	MS RAST MODE	HW Mode
NUMRASTSAMPLES_1	OFF_PIXEL	**Legacy Non-MSAA Mode** 1X rasterization, using Pixel Location Legacy lines or AA-line rasterization 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PIXEL	**1X Multisampling Mode** 1X rasterization, using Pixel Location MSAA lines only, using Pixel Location 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location
	ON_PATTERN	Invalid
	OFF_PATTERN	Invalid
n where n > NUMRASTSAMPLES_1	OFF_PIXEL	**Invalid**
	ON_PIXEL	**Invalid**
	OFF_PATTERN	**Mixed Mode, PerPixel PS** See Note B above. Non-Lines: nX rasterization, using Sample Offsets 1X PS, sample at Pixel Location 1X output merge, eval depth atPixel Location
	ON_PATTERN	**Pattern RTIR, PerPixel PS** nX rasterization, using Sample Offsets MSAA lines only 1X PS, sample at Pixel Location 1X output merge, eval Depth at Pixel Location

Note: Multisample Dispatch Mode is not taken into account in Table 4 given that with RTIR:

Details
The value of PERSAMPLE is converted to PERPIXEL internally.

Other WM Functions

The only other WM function is Statistics Gathering.

Statistics Gathering

If **Statistics Enable** is set in WM_STATE or 3DSTATE_WM, the Windower increments the PS_INVOCATIONS_COUNT register once for each unmasked pixel (or sample) that is *dispatched* to a Pixel Shader thread.

If **Early Depth Test Enable** is set it is possible for pixels or samples to be discarded before reaching the Pixel Shader due to failing the depth or stencil test. PS_INVOCATIONS_COUNT will still be incremented for these pixels or samples since the depth test occurs after the pixel shader from the point of view of SW.

Pixel

This section contains the following subsections:

- **Depth and Stencil**, which covers the Depth and Stencil test functions
- **Pixel Dispatch**, which covers pixel shader state, pixel grouping, multisampling effects on pixel shader dispatch, and pixel shader thread payload
- **Pixel Backend**, which covers backend processing

Early Depth/Stencil Processing

The Windower/IZ unit provides the Early Depth Test function, a major performance-optimization feature where an attempt is made to remove pixels that fail the Depth and Stencil Tests prior to pixel shading. This requires the WM unit to perform the interpolation of pixel ("source") depth values, read the current ("destination") depth values from the cached depth buffer, and perform the Depth and Stencil Tests As the WM unit has per-pixel source and destination Z values, these values are passed in the PS thread payload, if required.

Depth Offset

Note: The depth offset function is contained in SF unit, thus the state to control it is also contained in SF unit.

There are occasions where the Z position of some objects need to be slightly offset to reduce artifacts due to coplanar or near-coplanar primitives. A typical example is drawing the edges of triangles as wireframes – the lines need to be drawn slightly closer to the viewer to ensure they will not be occluded by the underlying polygon. Another example is drawing objects on a wall – without a bias on the z positions, they might be fully or partially occluded by the wall.

The device supports *global* depth offset, applied only to triangles, that bases the offset on the object's z slope Note that there is no clamping applied at this stage after the Z position is offset – clamping to [0,1] can be performed later after the Z position is interpolated to the pixel. This is preferable to clamping prior to interpolation, as the clamping would change the Z slope of the entire object.

The Global Depth Offset function is controlled by the **Global Depth Offset Enable** state variable in WM_STATE Global Depth Offset is only applied to 3DOBJ_TRIANGLE objects.

When Global Depth Offset Enable is ENABLED, the pipeline will compute:

MaxDepthSlope = max(abs(dZ/dX),abs(dz/dy)) // approximation of max depth slope for polygon

When UNORM Depth Buffer is at Output Merger (or no Depth Buffer):

```
Bias = GlobalDepthOffsetConstant * r + GlobalDepthOffsetScale * MaxDepthSlope
```

Where r is the minimum representable value > 0 in the depth buffer format, converted to float32 (note: If state bit **Legacy Global Depth Bias Enable** is set, the r term will be forced to 1.0)

When Floating Point Depth Buffer at Output Merger:

Bias = GlobalDepthOffsetConstant * 2^(exponent(max z in primitive) – r) + GlobalDepthOffsetScale * MaxDepthSlope

Where r is the # of mantissa bits in the floating point representation (excluding the hidden bit), e.g. 23 for float32 (note: If state bit Legacy Global Depth Bias Enable is set, no scaling is applied to the GobalDepthOffsetConstant).

Adding Bias to z:

```
if (GlobalDepthOffsetClamp > 0)
Bias = min(DepthBiasClamp, Bias)
else if(GlobalDepthOffsetClamp < 0)
Bias = max(DepthBiasClamp, Bias)
// else if GlobalDepthOffsetClamp == 0, no clamping occurs
z = z + Bias
```

Biasing is constant for a given primitive. The biasing formulas are performed with float32 arithmetic Global Depth Bias is not applied to any point or line primitives.

Early Depth Test/Stencil Test/Write

When **Early Depth Test Enable** is ENABLED, the WM unit will attempt to discard depth-occluded pixels during scan conversion (before processing them in the Pixel Shader). Pixels are only discarded when the WM unit can ensure that they would have no impact to the ColorBuffer or DepthBuffer. This function is therefore only a performance feature.

Note: For **[CHV, BSW]**, the **Early Depth Test Enable** bit is no longer present. This function is always enabled.

If some pixels within a subspan are discarded, only the pixel mask is affected indicating that the discarded pixels are not active. If all pixels within a subspan are discarded, that subspan will not even be dispatched.

Software-Provided PS Kernel Info

For the WM unit to properly perform Early Depth Test and supply the proper information in the PS thread payload (and even determine if a PS thread needs to be dispatched), it requires information regarding the PS kernel operation This information is provided by a number of state bits in WM_STATE, as summarized in the following table.

State Bit	Description
Pixel Shader Kill Pixel	This must be set when there is a chance that valid pixels passed to a PS thread may be discarded. This includes the discard of pixels by the PS thread resulting from a "killpixel" or "alphatest" function or as dictated by the results of the sampling of a "chroma-keyed" texture The WM unit needs this information to prevent early depth/stencil writes for pixels which might be killed by the PS thread, etc. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Computed Depth	This must be set when the PS thread computes the "source" depth value (i.e., from the API POV, writes to the "oDepth" output) In this case the WM unit can't make any decisions based on the WM-interpolated depth value. See WM_STATE/3DSTATE_WM for more information.
Pixel Shader Uses Source Depth	Must be set if the PS thread requires the WM-interpolated source depth value. This forces the source depth to be passed in the thread payload where otherwise the WM unit would not have seen it as required. See WM_STATE/3DSTATE_WM for more information.

Hierarchical Depth Buffer

A hierarchical depth buffer is supported to reduce memory traffic due to depth buffer accesses. This buffer is supported only in Tile Y memory.

The **Surface Type**, **Height**, **Width**, **Depth**, **Minimum Array Element**, **Render Target View Extent**, and **Depth Coordinate Offset X/Y** of the hierarchical depth buffer are inherited from the depth buffer. The height and width of the hierarchical depth buffer that must be allocated are computed by the following formulas, where HZ is the hierarchical depth buffer and Z is the depth buffer. The Z_Height, Z_Width, and Z_Depth values given in these formulas are those present in 3DSTATE_DEPTH_BUFFER incremented by one.

The value of Z_Height and Z_Width must each be multiplied by 2 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_4. The value of Z_Height must be multiplied by 2 and Z_Width must be multiplied by 4 before being applied to the table below if **Number of Multisamples** is set to NUMSAMPLES_8.

Surface Type	HZ_Width (Bytes)	HZ_Height (Rows)	HZ_Qpitch (Rows)
SURFTYPE_1D	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * Z_Depth	see below
SURFTYPE_2D	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * Z_Depth	see below
SURFTYPE_3D	ceiling(Z_Width / 16) * 16	see below	not applicable
SURFTYPE_CUBE	ceiling(Z_Width / 16) * 16	ceiling((HZ_QPitch/2)/8) *8 * 6 * Z_Depth	see below

To compute the minimum QPitch for the HZ surface, the height of each LOD in pixels is determined using the equations for hL in the GPU Overview volume, using a vertical alignment j=8. The following equation gives the minimum HZ_QPitch based on largest LOD m defined in the surface:

$$HZ\_QPitch = h_0 + max\left(h_1, \sum_{i=2}^{m} \mathrm{h_i}\right)$$

If m is less than 2, treat all h_L with L > m as zero and use the above equation.

The minimum HZ_Height required for a 3D surface must be computed based on h_L parameters documented in the GPU Overview volume, and the maximum LOD m:

$$HZ\_Height = \frac{1}{2}\left[\sum_{i=0}^{m} h_i * floor\left(\frac{Z\_Depth}{2^i}\right)\right]$$

The format of the data in the hierarchical depth buffer is not documented here, as this surface needs only to be allocated by software. Hardware will read and write this surface during operation and its contents are discarded once the last primitive is rendered that uses the hierarchical depth buffer.

The hierarchical depth buffer can be enabled whenever a depth buffer is defined, with its effect being invisible other than generally higher performance. The only exception is the hierarchical depth buffer must be disabled when using software tiled rendering.

If HiZ is enabled, you must initialize the clear value by either:

1. Perform a depth clear pass to initialize the clear value.
2. Send a 3dstate_clear_params packet with valid = 1.

Without one of these events, context switching will fail, as it will try to save off a clear value even though no valid clear value has been set. When context restore happens, HW will restore an uninitialized clear value.

Depth Buffer Clear

With the hierarchical depth buffer enabled, performance is generally improved by using the special clear mechanism described here to clear the hierarchical depth buffer and the depth buffer. This is enabled though the **Depth Buffer Clear** field in WM_STATE or 3DSTATE_WM or using the 3DSTATE_WM_HZ_OP. This bit can be used to clear the depth buffer in the following situations:

- Complete depth buffer clear.
- Partial depth buffer clear with the clear value the same as the one used on the previous clear.
- Partial depth buffer clear with the clear value different than the one used on the previous clear can use this mechanism if a depth buffer resolve is performed first.

The following is required when performing a depth buffer clear using any of the above clearing methods (WM_STATE, 3DSTATE_WM or 3DSTATE_WM_HZ_OP).

- The hierarchical depth buffer enable must be set in the 3DSTATE_DEPTH_BUFFER.
- The fields in 3DSTATE_CLEAR_PARAMS are set to indicate the source of the clear value and (if source is in this command) the clear value itself.
- The clear value must be between the min and max depth values (inclusive) defined in the CC_VIEWPORT. If the depth buffer format is D32_FLOAT, then +/-DENORM values are also allowed.
    - The following alignment restrictions need to be met while doing the fast-clear:

Alignment Restriction
The following restrictions apply only if the depth buffer surface type is D16_UNORM and software does not use the "full surf clear": If **Number of Multisamples** is NUMSAMPLES_1, the rectangle must be aligned to an 8x4 pixel block relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all 8x4 pixels must be lit. If **Number of Multisamples** is NUMSAMPLES_2, the rectangle must be aligned to a 4x4 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 4x4 pixels must be lit. If **Number of Multisamples** is NUMSAMPLES_4, the rectangle must be aligned to a 4x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 4x2 pixels must be lit. If **Number of Multisamples** is NUMSAMPLES_8, the rectangle must be aligned to a 2x2 pixel block (8x4 sample block) relative to the upper left corner of the depth buffer, and contain an integer number of these pixel blocks, and all samples of the 2x2 pixels must be lit.
The minimum granularity of clear is one pixel, but all samples of the pixel must be cleared. Clearing partial samples of a pixel is not supported. If a newly allocated depth buffer is not 8x4 pixel aligned (if MSAA enabled, 4x4 for 2X, 4x2 for 4X, 2x2 for 8X), and if the first operation on the depth buffer does not clear the entire width and height of the surface, then first a HiZ ambiguate must be done on the portions of the depth buffer that are not cleared. If the depth buffer clear operation does

Alignment Restriction
clear the entire width and height of the surface, then the "full surface clear" bit in 3DSTATE_WM_OP must be set to 1.

The following is required when performing a depth buffer clear with using the WM_STATE or 3DSTATE_WM:

- If other rendering operations have preceded this clear, a PIPE_CONTROL with depth cache flush enabled, Depth Stall bit enabled must be issued before the rectangle primitive used for the depth buffer clear operation.
- **Depth Test Enable** must be disabled and **Depth Buffer Write Enable** must be enabled (if depth is being cleared).
- Stencil buffer clear can be performed at the same time by enabling Stencil Buffer Write Enable. Stencil Test Enable must be enabled and Stencil Pass Depth Pass Op set to REPLACE, and the clear value that is placed in the stencil buffer is the **Stencil Reference Value** from COLOR_CALC_STATE.
- Note also that stencil buffer clear can be performed without depth buffer clear. For stencil only clear, **Depth Test Enable** and **Depth Buffer Write Enable** must be disabled.

In some cases **Depth Buffer Clear** cannot be enabled and the legacy method of clearing must be used:

- If the depth buffer format is D32_FLOAT_S8X24_UINT or D24_UNORM_S8_UINT.
- If stencil test is enabled but the separate stencil buffer is disabled.

Depth buffer clear pass using any of the methods (WM_STATE, 3DSTATE_WM or 3DSTATE_WM_HZ_OP) must be followed by a PIPE_CONTROL command with DEPTH_STALL bit and Depth FLUSH bits "**set**" before starting to render. DepthStall and DepthFlush are not needed between consecutive depth clear passes nor is it required if the depth-clear pass was done with "full_surf_clear" bit set in the 3DSTATE_WM_HZ_OP.

Note: If using the optimized depth buffer clear, this pipecontrol should be done after the resetting of the clear/resolve bits in the 3DSTATE_WM_HZ_OP (step #8).

Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete If the depth buffer is retained and used for another purpose (i.e as input to the sampling engine as a shadow map), it must first be "resolved" This is done by setting the **Depth Buffer Resolve Enable** field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the nest rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

The following is required when performing a depth buffer resolve:

- The surface must have been initialized with a Depth Buffer Clear after its allocation to initialize the Depth Clear Value.
- A rectangle primitive of the same size as the previous depth buffer clear operation must be delivered, and depth buffer state cannot have changed since the previous depth buffer clear operation.
- **Depth Test Enable** must be enabled with the **Depth Test Function** set to NEVER. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel** and **Pixel Shader Computed Depth** must all be disabled.

Programming Note	
Context:	HTML
HW uses the clear value from the 3DSTATE_CLEAR_PARAM. If you change the value in the 3DSTATE_CLEAR_PARAMS before resolve, it will flush the depth caches and have the new-clear value in its register. When doing the resolve pass, it is driver's responsibility to make sure that the clear-value for the depth buffer is the same one as the clear-pass.	

Hierarchical Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation If this occurs, the hierarchical depth buffer must be *resolved* to avoid incorrect device behavior. This is done by setting the Hierarchical Depth Buffer Resolve Enable field in WM_STATE or 3DSTATE_WM and rendering a full render target sized rectangle. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be delivered.
- **Depth Test Enable** must be disabled. **Depth Buffer Write Enable** must be enabled. **Stencil Test Enable** and **Stencil Buffer Write Enable** must be disabled.
- **Pixel Shader Dispatch, Alpha Test, Pixel Shader Kill Pixel**, and **Pixel Shader Computed Depth** must all be disabled.

Optimized Depth Buffer Clear and/or Stencil Buffer Clear

With the hierarchical depth buffer enabled, performance is generally improved by using the special clear mechanism described here to clear the hierarchical depth buffer and the depth buffer This is enabled though the **Depth Buffer Clear** field in 3DSTATE_WM_HZ_OP This bit can be used to clear the depth buffer in the following situations:

- All 3D units before SF will be bypassed by WM_HZ_OP and states for those units need not be set/restored for these rectangles.
- Complete depth buffer clear
- Partial depth buffer clear with the clear value the same as the one used on the previous clear
- Partial depth buffer clear with the clear value different than the one used on the previous clear can use this mechanism if a depth buffer resolve is performed first.
- The minimum granularity of clear is one pixel, but all samples of the pixel must be cleared. Clearing partial samples of a pixel is not supported

Stencil Buffer Clears can be alone or at the same time as depth buffer clears by using the Stencil Buffer Clear bit in 3DSTATE_WM_HZ_OP.

The proper sequence of commands is as follows:

1. Setup 3DSTATE_DEPTH_BUFFER (as needed). Render Target Array index will be internally force to zero. SW must set 3DSTATE_DEPTH_BUFFER::**MinimumArrayElement** to render to the array to be cleared.
2. Setup 3DSTATE_HIER_DEPTH_BUFFER (as needed)
3. Setup 3DSTATE_STENCIL_BUFFER (as needed)
4. Setup 3DSTATE_CLEAR_PARMS (as needed)
5. Setup 3DSTATE_DRAWING_RECTANGLE (as needed and only if it is different from already existing drawing rectangle)
6. 3DSTATE_WM_HZ_OP w/ 1 of the clear/resolve bits set
   // This overrides existing state and forces them to what is needed for the clear
   // This also carries the vertex info for doing the clear
7. PIPE_CONTROL w/ all bits clear except for “Post-Sync Operation” must set to “Write Immediate Data” enabled.
   // This causes 3DSTATE_WM_HZ_OP state to be committed to SF and WM as a pipeline state. Once state is committed to SF, causes to spawn a rectangle to be drawn
8. 3DSTATE_WM_HZ_OP w/ none of the clear/resolve bits set
   // This clears the overrides
9. Restore 3DSTATE_DEPTH_BUFFER (as needed).
10. Restore 3DSTATE_HIER_DEPTH_BUFFER (as needed)
11. Restore 3DSTATE_STENCIL_BUFFER (as needed)

Arbitrary size rectangles are supported using the Top Left X, Top Left Y, Bottom Right X, Bottom Right Y fields in the 3DSTATE_WM_HZ_OP.

Optimized Depth Buffer Resolve

If the hierarchical depth buffer is enabled, the depth buffer may contain incorrect results after rendering is complete If the depth buffer is retained and used for another purpose (i.e locked by the app), it must first be "resolved" This is done by setting the **Depth Buffer Resolve Enable** field in 3DSTATE_WM_HZ_OP. The depth buffer resolve uses the same sequence as the optimized Depth buffer clear (see above) except the **Depth Buffer Resolve Enable** bit is set. Once this is complete, the depth buffer will contain the same contents as it would have had the rendering been performed with the hierarchical depth buffer disabled. In a typical usage model, depth buffer needs to be resolved after rendering on it and before using a depth buffer as a source for any consecutive operation. Depth buffer can be used as a source in three different cases: using it as a texture for the nest rendering sequence, honoring a lock on the depth buffer to the host OR using the depth buffer as a blit source.

Doing a resolve operation requires that a preceding Depth Buffer Clear operation is required to have initialized the Depth Clear Value.

Optimized Hierarchical Depth Buffer Resolve

For CHV, BSW, if the hierarchical depth buffer is enabled, the hierarchical depth buffer may contain incorrect results if the depth buffer is written to outside of the 3D rendering operation. If this occurs, the hierarchical depth buffer must be "resolved" to avoid incorrect device behavior. This is done by setting the **Hierarchical Depth Buffer Resolve Enable** field in 3DSTATE_WM_HZ_OP and specifying a full render target sized rectangle. The depth buffer resolve uses the same sequence as the optimized Depth buffer clear (see above) except the **Hierarchical Depth Buffer Resolve Enable** bit is set. Once this is complete, the hierarchical depth buffer will contain contents such that rendering will give the same results as it would have had the rendering been performed with the hierarchical depth buffer disabled.

The following is required when performing a hierarchical depth buffer resolve:

- A rectangle primitive covering the full render target must be programmed on Xmin, Ymin, Xmax, and Ymax in the 3DSTATE_WM_HZ_OP command.
- The rectangle primitive size must be aligned to 8x4 pixels.

Separate Stencil Buffer

The following tables describe the separate stencil buffer for different generations.

The separate stencil buffer is always enabled, thus the field in 3DSTATE_DEPTH_BUFFER to explicitly enable the separate stencil buffer has been removed. Surface formats with interleaved depth and stencil are no longer supported.

The stencil buffer has a format of R8_UNIT, and shares **Surface Type**, **Height**, **Width**, and **Depth**, **Minimum Array Element**, **Render Target View Extent**, **Depth Coordinate Offset X/Y**, **LOD**, and **Depth Buffer Object Control State** fields of the depth buffer.

DepthStencil Buffer State

This section contains the state registers for the Depth/Stencil Buffers.

3DSTATE_DEPTH_BUFFER

3DSTATE_STENCIL_BUFFER

3DSTATE_HIER_DEPTH_BUFFER

3DSTATE_CLEAR_PARAMS

Pixel Shader Thread Generation

After a group of object fragments have been rasterized, the Pixel Shader (PSD) function is invoked to further compute output information and cause results to be written to output surfaces (like color, depth, stencil, UAvs etc). Fragments can be P or S.

For each fragment, the Pixel Shader calculates the values of the various vertex attributes that are to be interpolated across the object using the interpolation coefficients. It then executes an API-supplied Pixel Shader Program. Instructions in this program permit the accessing of texture map data, where Texture Samplers are employed to sample and filter texture maps (see the Shared Functions chapter). Arithmetic operations can be performed on the texture data, input fragment information, and Pixel Shader Constants to compute the resultant fragment's output. The Pixel Shader program also allows the pixel to be discarded from further processing.

3DSTATE_PS

This command is used to set state used by the pixel shader dispatch stage.

3DSTATE_PS

Programming Note	
Context:	Pixel Shader Thread Generation
Note: The PS Unit also receives 3DSTATE_PS_BLEND, 3DSTATE_SAMPLEMASK, 3DSTATE_MULTISAMPLE, and 3DSTATE_PS_EXTRA.	

Signal [CHV, BSW]	PS_INT::oMask Present to RenderTarget
Description	This bit is inserted in the PS payload header and made available to the DataPort (either via the message header or via header bypass) to indicate that oMask data (one or two phases) is included in Render Target Write messages. If present, the oMask data is used to mask off samples.
Formula	= 3DSTATE_PS_EXTRA::oMask Present to RenderTarget

Signal [CHV, BSW]	PS_INT::Dual Source Blend Enable
Description	This field is set if dual source blend is enabled. If this bit is disabled, the data port dual source message reverts to a single source message using source 0.
Formula	= 3DSTATE_PS_BLEND::ColorBufferBlendEnable && (PS_INT::UsesSrc1BlendFactor || (PS_INT::IndependentAlphaUsesSrc1BlendFactors && 3DSTATE_PS_BLEND::Independent Alpha Blend Enable))

Signal [CHV, BSW]	PS_INT::UsesSrc1BlendFactor
Description	
Formula	= (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_SRC1_ALPHA) ||

	(3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA)

Signal [CHV, BSW]	PS_INT::IndependentAlphaUsesSrc1BlendFactors
Description	
Formula	= (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::SourceAlphaBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_SRC1_ALPHA) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_INV_SRC1_COLOR) || (3DSTATE_PS_BLEND::DestinationAlphaBlendFactor == BLENDFACTOR_INV_SRC1_ALPHA)

Signal [CHV, BSW]	PS_INT::PS UAV-only
Description	Pixel Shader UAV-only render target
Formula	= 3DSTATE_PS_EXTRA::Pixel Shader Has UAV && !3DSTATE_PS_EXTRA:: Pixel Shader Does not write to RT

3DSTATE_PS_EXTRA

This command is used to set state used by the pixel shader dispatch stage.

3DSTATE_PS_BLEND

This command is used to set state used by the pixel shader dispatch stage.

3DSTATE_CONSTANT_PS

3DSTATE_BINDING_TABLE_POINTERS_PS

3DSTATE_PUSH_CONSTANT_ALLOC_PS

3DSTATE_SAMPLER_STATE_POINTERS_PS

Pixel Grouping (Dispatch Size) Control

The WM unit can pass a grouping of 2 subspans (8 pixels), 4 subspans (16 pixels), or 8 subspans (32 pixels) to a Pixel Shader thread. Software should take into account the following considerations when determining which groupings to support/enable during operation. This determination involves a tradeoff of these likely conflicting issues. Note that the size of the dispatch has significant impact on the kernel program. (It is certainly not transparent to the kernel.) Also note that there is no implied spatial relationship between the subspans passed to a PS thread, other than the fact that they come from the same object.

- **Thread Efficiency**: In general, there is some amount of overhead involved with PS thread dispatch, and if this can be amortized over a larger number of pixels, efficiency will likely increase. This is especially true for very short PS kernels, as may be used for desktop composition, etc.
- **GRF Consumption**: Processing more pixels per thread requires a larger thread payload and likely more temporary register usage, both of which translate into a requirement for a larger GRF register allocation for the threads. This increased GRF usage could lead to increased use of scratch space (for spill/fill, etc.) and possibly less efficient use of the EUs (as it would be less likely to find an EU with enough free physical GRF registers to service the thread).
- **Object Size**: If the number of very small objects (e.g., covering 2 subspans or fewer) is expected to comprise a significant portion of the workload, supporting the 8-pixel dispatch mode may be advantageous. Otherwise there could be a large number of 16-pixel dispatches with only 1 or 2 valid subspans, resulting in low efficiency for those threads.
- **Intangibles**: Kernel footprint & Instruction Cache impact; Complexity; ….

The groupings of subspans that the WM unit is allowed to include in a PS thread payload is controlled by the **32,16,8 Pixel Dispatch Enable** state variables programmed in WM_STATE. Using these state variables, the WM unit attempts to dispatch the largest allowed grouping of subspans. The following table lists the possible combinations of these state variables.

Please note that, the valid column in the table indicates which products supports the combination dispatch. Combinations that are not listed in the table are not available on any product.

The letter codes A, B, D, and E used in the Variable Pixel Dispatch table below are valid for all projects with some specific mode restrictions for specific projects for B, D, and E as indicated in the next few tables.

D is like B with an added general restriction, that it cannot be used in non-1x PERSAMPLE mode.

E cannot be used in PERSAMPLE mode with number of multisamples >= 2.

Variable Pixel Dispatch

Contiguous 64 Pixel Dispatch Enable	Contiguous 32 Pixel Dispatch Enable	32 Pixel Dispatch Enable	16 Pixel Dispatch Enable	8 Pixel Dispatch Enable	Valid	IP for n-pixel Contiguous Dispatch		IP for n-pixel Dispatch (KSP offsets are in 128-bit instruction units)		
						n=64	n=32	n=32	n=16	n=8
0	0	0	0	1	A					KSP[0]
0	0	0	1	0	B				KSP[0]	
0	0	0	1	1	D				KSP[2]	KSP[0]
0	0	1	0	0	B			KSP[0]		
0	0	1	1	0	E			KSP[1]	KSP[2]	
0	0	1	1	1	D			KSP[1]	KSP[2]	KSP[0]
0	1	1	1	0	D		KSP[2]	KSP[1]	KSP[0]	
1	0	1	1	0	D	KSP[2]		KSP[1]	KSP[0]	

Each of the three KSP values is separately specified. In addition, each kernel has a separately-specified GRF register count.

Depending on the subspan grouping selected, the WM unit will modify the starting PS Instruction Pointer (derived from the Kernel Start Pointer in WM_STATE) as a means to inform the PS kernel of the number of subspans included in the payload. The modified IP is a function of the enabled modes and the dispatch size, as shown in the table below.

The driver must ensure that the PS kernel begins with a corresponding jump table to properly handle the number of subspans dispatched. The WM unit will "OR" in the two LSBs of the Kernel Pointer (bits 5:4) to create an instruction level address. (Note that the pointer from WM_STATE is 64-byte aligned which corresponds to four 128-bit instructions.)

If only one dispatch mode is enabled, the Jitter should not include any jump table entries at the beginning of the PS kernel. If multiple dispatch modes are enabled, a two entry jump table should always be inserted, regardless of which modes are enabled (jump table entry for 8 pixel dispatch, followed by jump table entry for 32 pixel dispatch).

Note that for SIMD32 dispatch, pixel shader dispatch function increments GRF Start Register for URB Data state by 2 to account for the additional SIMD16 payload. The Pixel Shader kernel needs to comprehend this modification for SIMD32.

```
if ( 32PixelDispatchEnable && n > 7 )
    Dispatch 32 Pixels
else if ( 16PixelDispatchEnable && ( n > 2 || ! 8PixelDispatchEnable) )
    Dispatch 16 Pixels
else
    Dispatch 8 Pixels
end if
```

Multisampling Effects on Pixel Shader Dispatch

The pixel shader payloads are defined in terms of subspans and pixels. The slots in the pixel shader thread previously mapped 1:1 with pixels. With multisampling, a slot could contain a pixel or may just contain a single sample, depending on the mode. Payload definitions now refer to *slot* to make the definition independent of multisampling mode.

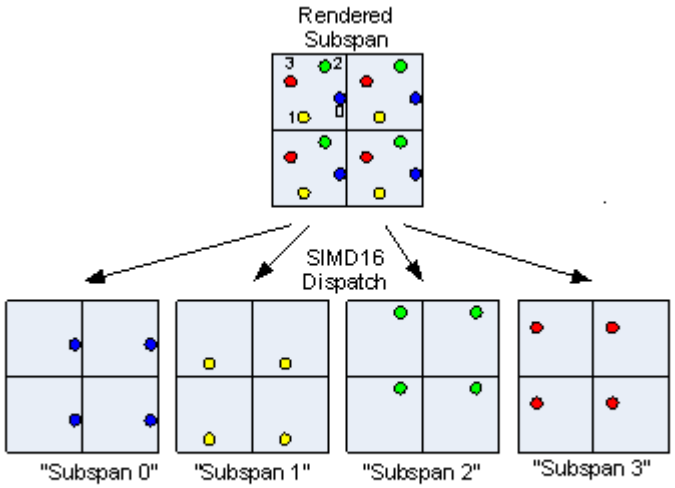
MSDISPMODE_PERPIXEL Thread Dispatch

In PERPIXEL mode, the pixel shader kernel still works on 2/4/8 separate subspans, depending on dispatch mode. The fact that rasterization and the depth/stencil tests are being performed on a per-sample (not per-pixel) basis is transparent to the pixel shader kernel.

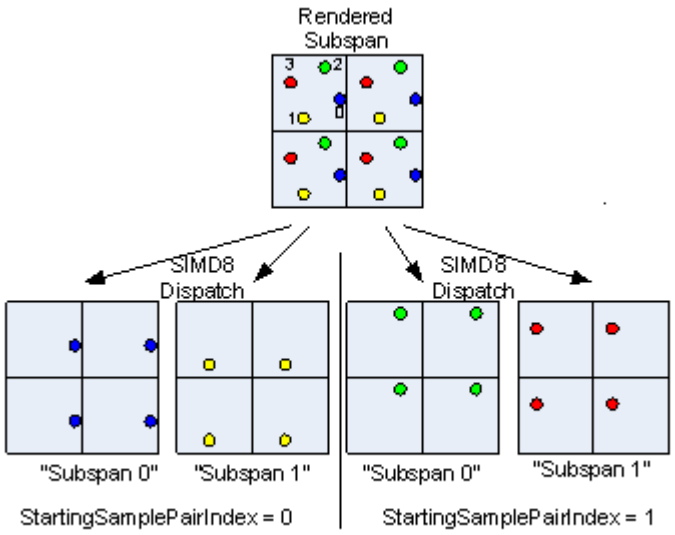
MSDISPMODE_PERSAMPLE Thread Dispatch

In PERSAMPLE mode, the pixel shader needs to operate on a sample vs. pixel basis (although this collapses in NUMSAMPLES_1 mode) Instead of processing strictly different subspans in parallel , the PS kernel processes different sample indices of one or more subspans in parallel For example, a SIMD16 dispatch in PERSAMPLE/NUMSAMPLES_4 mode would operate on a single subspan, with the usual "4 Subspan0 pixel slots" used for the "4 Sample0 locations of the (single) subspan" Subspan1 slots would be used for the Sample1 locations, and so on This layout allows the pixel shader to compute derivatives/LOD based on deltas between corresponding sample locations in the subspan in the same fashion as LEGACY pixel shader execution, and as required by DX10.1.

Depending on the dispatch mode (8/16/32 pixels) and multisampling mode (1X/4X), there are different mappings of subspans/samples onto dispatches and slots-within-dispatch In some cases, more than one subspan may be included in a dispatch, while in other cases multiple dispatches are be required to process all samples for a single subspan In the latter case, the **StartingSamplePairIndex** value is included in the payload header so the Render Target Write message will access the correct samples with each message.



PERSAMPLE SIMD16 4X Dispatch



PERSAMPLE Dispatch

The following table provides the complete dispatch/slot mappings for all the MS/Dispatch combinations.

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
SIMD32	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]* *Slot[19:16] = Subspan[4].Pixel[3:0].Sample[0]* *Slot[23:20] = Subspan[5].Pixel[3:0].Sample[0]* *Slot[27:24] = Subspan[6].Pixel[3:0].Sample[0]* *Slot[31:28] = Subspan[7].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]* *Slot[19:16] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[23:20] = Subspan[2].Pixel[3:0].Sample[1]* *Slot[27:24] = Subspan[3].Pixel[3:0].Sample[0]*

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
		Slot[31:28] = Subspan[3].Pixel[3:0].Sample[1]
	4X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]* *Slot[19:16] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[23:20] = Subspan[1].Pixel[3:0].Sample[1]* *Slot[27:24] = Subspan[1].Pixel[3:0].Sample[2]* *Slot[31:28] = Subspan[1].Pixel[3:0].Sample[3]*
	8X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]* *Slot[19:16] = Subspan[0].Pixel[3:0].Sample[4]* *Slot[23:20] = Subspan[0].Pixel[3:0].Sample[5]* *Slot[27:24] = Subspan[0].Pixel[3:0].Sample[6]* *Slot[31:28] = Subspan[0].Pixel[3:0].Sample[7]*
SIMD16	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[11:8] = Subspan[2].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[3].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[1].Pixel[3:0].Sample[0]* *Slot[15:12] = Subspan[1].Pixel[3:0].Sample[1]*
	4X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[3]*

Dispatch Size	Num Samples	Slot Mapping (SSPI = Starting Sample Pair Index)
	8X	*Dispatch[i]: (i=0, 2)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]* *Slot[11:8] = Subspan[0].Pixel[3:0].Sample[SSPI*2+2]* *Slot[15:12] = Subspan[0].Pixel[3:0].Sample[SSPI*2+3]*
SIMD8	*1X*	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[1].Pixel[3:0].Sample[0]*
	2X	*Slot[3:0] = Subspan[0].Pixel[3:0].Sample[0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[1]*
	4X	*Dispatch[i]: (i=0..1)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]*
	8X	*Dispatch[i]: (i=0, 1, 2, 3)* *SSPI = i* *Slot[3:0] = Subspan[0].Pixel[3:0].Sample[SSPI*2+0]* *Slot[7:4] = Subspan[0].Pixel[3:0].Sample[SSPI*2+1]*

PS Thread Payload for Normal Dispatch

The following table lists all possible contents included in a PS thread payload, in the order they are provided. Certain portions of the payload are optional, in which case the corresponding phase is skipped.

This payload does not apply to the contiguous dispatch modes. The payload for these modes is documented in the section titled *PS Thread Payload for Contiguous Dispatch*.

PS Thread Payload for Normal Dispatch

The following payload table applies to **CHV, BSW**. All registers are numbered starting at 0, but many registers are skipped depending on configuration. This causes all registers below to be renumbered to fill in the skipped locations. The only case where actual registers may be skipped is immediately before the constant data and again before the setup data.

PS Thread Payload for Normal Dispatch [CHV, BSW]

DWord	Bits	Description
R0.7	31	Reserved.
	30:24	Reserved
	23:0	**Primitive Thread ID:** This field contains the primitive thread count passed to the Windower from the Strips Fans Unit. Format: Reserved for HW Implementation Use.
R0.6	31:24	Reserved
	23:0	**Thread ID:** This field contains the thread count which is incremented by the Windower for every thread that is dispatched. Format: Reserved for HW Implementation Use.
R0.5	31:10	**Scratch Space Pointer:** Specifies the 1K-byte aligned pointer to the scratch space available for this PS thread. This is specified as an offset to the **General State Base Address**. Format = GeneralStateOffset[31:10]
	9:8	Reserved
	7:0	**FFTID:** This ID is assigned by the WM unit and is an identifier for the thread. It is used to free up resources used by the thread upon thread completion. Format: Reserved for HW Implementation Use.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved
R0.3	31:5	**Sampler State Pointer:** Specifies the 32-byte aligned pointer to the Sampler State table. It is specified as an offset from the **Dynamic State Base Address**. Format = DynamicStateOffset[31:5]
	4	Reserved
	3:0	**Per Thread Scratch Space:** Specifies the amount of scratch space allowed to be used by this thread. **Programming Notes:** This amount is available to the kernel for information only. It will be passed verbatim (if not altered by the kernel) to the Data Port in any scratch space access messages, but the Data Port will ignore it. Format = U4

DWord	Bits	Description
		Range = [0,11] indicating [1k bytes, 2M bytes] in powers of two
R0.2	31:0	Reserved: Delivered as zeros (reserved for message header fields).
R0.1	31:6	**Color Calculator State Pointer:** Specifies the 64-byte aligned pointer to the Color Calculator state (COLOR_CALC_STATE structure in memory). It is specified as an offset from the **Dynamic State Base Address**. This value is eventually passed to the ColorCalc function in the DataPort and is used to fetch the corresponding CC_STATE data. Format = DynamicStateOffset[31:5]
	5:0	Reserved
R0.0	31	Reserved
	30:27	**Viewport Index:** Specifies the index of the viewport currently being used. Format = U4 Range = [0,15]
	26:16	**Render Target Array Index:** Specifies the array index to be used for the following surface types: SURFTYPE_1D: specifies the array index Range = [0,2047] SURFTYPE_2D: specifies the array index Range = [0,2047] SURFTYPE_3D: specifies the "r" coordinate Range = [0,2047] SURFTYPE_CUBE: specifies the face identifier Range = [0,5] Face / Render Target Array Index: +x 0; -x 1; +y 2; -y 3; +z 4; -z 5 Format = U11
	15	**Front/Back Facing Polygon:** Determines whether the polygon is front or back facing. Used by the render cache to determine which stencil test state to use. 0: Front Facing 1: Back Facing
	14	Reserved
	13	**Source Depth to Render Target:** Indicates that source depth will be sent to the render target.
	12	**oMask to Render Target:** Indicates that oMask will be sent to the render target.

DWord	Bits	Description
	11:9	Reserved
	8	Reserved for expansion of **Starting Sample Pair Index**.
	7:6	**Starting Sample Pair Index:** Indicates the index of the first sample pair of the dispatch. Format = U2 Range = [0,3]
	5	Reserved
	4:0	**Primitive Topology Type:** This field identifies the Primitive Topology Type associated with the primitive spawning this object. The WM unit does not modify this value (e.g., objects within POINTLIST topologies see POINTLIST). Format: (See 3DPRIMITIVE command in *3D Pipeline*.)
R1.7	31:16	**Pixel/Sample Mask (SubSpan[3:0]):** Indicates which pixels within the four subspans are lit. If 32 pixel dispatch is enabled, this field contains the pixel mask for the first four subspans. **Note:** This is not a duplicate of the Dispatch Mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[3:0]):** This is a duplicate copy of the pixel mask. This copy can be modified as the pixel shader thread executes in order to turn off pixels based on kill instructions.
R1.6	31:0	Reserved
R1.5	31:16	**Y3:** Y coordinate (screen space) for upper-left pixel of subspan 3 (slot 12). Format = U16
	15:0	**X3:** X coordinate (screen space) for upper-left pixel of subspan 3 (slot 12). Format = U16
R1.4	31:16	**Y2:** Y coordinate (screen space) for upper-left pixel of subspan 2 (slot 8). Format = U16
	15:0	**X2:** X coordinate (screen space) for upper-left pixel of subspan 2 (slot 8). Format = U16
R1.3	31:16	**Y1:** Y coordinate (screen space) for upper-left pixel of subspan 1 (slot 4). Format = U16

DWord	Bits	Description
	15:0	**X1:** X coordinate (screen space) for upper-left pixel of subspan 1 (slot 4). Format = U16
R1.2	31:16	**Y0:** Y coordinate (screen space) for upper-left pixel of subspan 0 (slot 0). Format = U16
	15:0	**X0:** X coordinate (screen space) for upper-left pixel of subspan 0 (slot 0). Format = U16
R1.1	31:0	Reserved
R1.0	31:20	Reserved
	15:12	Slot 3 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	11:8	Slot 2 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	7:4	Slot 1 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
	3:0	Slot 0 SampleID (if pixel or sample dispatch) Format = U4 1X MSAA range: [0]

DWord	Bits	Description
		2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]
		R2: Delivered only if this is a *32-pixel dispatch*.
R2.7	31:16	**Pixel/Sample Mask (SubSpan[7:4]):** Indicates which pixels within the upper four subspans are lit. This field is valid only when the 32 pixel dispatch state is enabled. This field must not be modified by the pixel shader thread. Note: This is not a duplicate of the dispatch mask that is delivered to the thread. The dispatch mask has all pixels within a subspan as active if any of them are lit to enable LOD calculations to occur correctly. This field must not be modified by the Pixel Shader kernel.
	15:0	**Pixel/Sample Mask Copy (SubSpan[7:4]):** This is a duplicate copy of pixel mask for the upper 16 pixels. This copy will be modified as the pixel shader thread executes to turn off pixels based on kill instructions.
R2.6	31:0	Reserved
R2.5	31:16	**Y7:** Y coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
	15:0	**X7:** X coordinate (screen space) for upper-left pixel of subspan 7 (slot 28) Format = U16
R2.4	31:16	Y6
	15:0	X6
R2.3	31:16	Y5
	15:0	X5
R2.2	31:16	Y4
	15:0	X4
R2.1	31:0	Reserved
R2.0	31:16	Reserved
	15:12	Slot 7 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7]

DWord	Bits	Description
		16X MSAA range [0..15]
	11:8	Slot 6 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	7:4	Slot 5 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
	3:0	Slot 4 SampleID Format = U4 1X MSAA range: [0] 2X MSAA range [0,1] 4X MSAA range [0..3] 8X MSAA range [0..7] 16X MSAA range [0..15]
		R3-R26: Delivered only if the corresponding **Barycentric Interpolation Mode** bit is set. Register phases containing Slot 8-15 data are not delivered in *8-pixel dispatch* mode.
R3.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 7 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R3.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 6
R3.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 5
R3.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 4

DWord	Bits	Description
R3.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 3
R3.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 2
R3.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 1
R3.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 0
R4		Perspective Pixel Location Barycentric[2] for Slots 7:0
R5.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 15
R5.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 14
R5.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 13
R5.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 12
R5.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 11
R5.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 10
R5.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 9
R5.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 8
R6		Perspective Pixel Location Barycentric[2] for Slots 15:8
R7:10		Perspective Centroid Barycentric
R11:14		Perspective Sample Barycentric
R15:18		Linear Pixel Location Barycentric
R19:22		Linear Centroid Barycentric
R23:26		Linear Sample Barycentric
		R27: Delivered only if **Pixel Shader Uses Source Depth** is set.
R27.7	31:0	Interpolated Depth for Slot 7 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth** (WM_STATE) is set.
R27.6	31:0	Interpolated Depth for Slot 6
R27.5	31:0	Interpolated Depth for Slot 5
R27.4	31:0	Interpolated Depth for Slot 4
R27.3	31:0	Interpolated Depth for Slot 3
R27.2	31:0	Interpolated Depth for Slot 2
R27.1	31:0	Interpolated Depth for Slot 1
R27.0	31:0	Interpolated Depth for Slot 0
		R28: Delivered only if **Pixel Shader Uses Source Depth** is set and this is not an *8-pixel dispatch*.
R28.7	31:0	Interpolated Depth for Slot 15
R28.6	31:0	Interpolated Depth for Slot 14
R28.5	31:0	Interpolated Depth for Slot 13
R28.4	31:0	Interpolated Depth for Slot 12

DWord	Bits	Description
R28.3	31:0	Interpolated Depth for Slot 11
R28.2	31:0	Interpolated Depth for Slot 10
R28.1	31:0	Interpolated Depth for Slot 9
R28.0	31:0	Interpolated Depth for Slot 8
		R29: Delivered only if **Pixel Shader Uses Source W** is set.
R29.7	31:0	Interpolated W for Slot 7 Format = IEEE_Float This and the next register phase are only included if **Pixel Shader Uses Source W** (WM_STATE) is set.
R29.6	31:0	Interpolated W for Slot 6
R29.5	31:0	Interpolated W for Slot 5
R29.4	31:0	Interpolated W for Slot 4
R29.3	31:0	Interpolated W for Slot 3
R29.2	31:0	Interpolated W for Slot 2
R29.1	31:0	Interpolated W for Slot 1
R29.0	31:0	Interpolated W for Slot 0
		R30: Delivered only if **Pixel Shader Uses Source W** is set and this is not an *8-pixel dispatch*.
R30.7	31:0	Interpolated W for Slot 15
R30.6	31:0	Interpolated W for Slot 14
R30.5	31:0	Interpolated W for Slot 13
R30.4	31:0	Interpolated W for Slot 12
R30.3	31:0	Interpolated W for Slot 11
R30.2	31:0	Interpolated W for Slot 10
R30.1	31:0	Interpolated W for Slot 9
R30.0	31:0	Interpolated W for Slot 8
		R31: Delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE.
R31.7	31:24	Position Offset Y for Slot 15 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 15 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state

DWord	Bits	Description
		of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 14
	7:0	Position Offset X for Slot 14
R31.6	31:24	Position Offset Y for Slot 13
	23:16	Position Offset X for Slot 13
	15:8	Position Offset Y for Slot 12
	7:0	Position Offset X for Slot 12
R31.5:4		Position Offset X/Y for Slot[11:8]
R31.3:2		Position Offset X/Y for Slot[7:4]
R31.1:0		Position Offset X/Y for Slot[3:0]
		R32: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set.
R32.7	31:0	Input Coverage Mask for Slot 7 Format = U32 This and the next register phase is only included if **Pixel Shader Uses Input Coverage Mask** (3DSTATE_PS) is set. This field always encodes sample Coverage Mask.
R32.6	31:0	Input Coverage Mask for Slot 6
R32.5	31:0	Input Coverage Mask for Slot 5
R32.4	31:0	Input Coverage Mask for Slot 4
R32.3	31:0	Input Coverage Mask for Slot 3
R32.2	31:0	Input Coverage Mask for Slot 2
R32.1	31:0	Input Coverage Mask for Slot 1
R32.0	31:0	Input Coverage Mask for Slot 0
		R33: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set and this is not an *8-pixel dispatch*.
R33.7	31:0	Input Coverage Mask for Slot 15
R33.6	31:0	Input Coverage Mask for Slot 14
R33.5	31:0	Input Coverage Mask for Slot 13
R33.4	31:0	Input Coverage Mask for Slot 12
R33.3	31:0	Input Coverage Mask for Slot 11
R33.2	31:0	Input Coverage Mask for Slot 10
R33.1	31:0	Input Coverage Mask for Slot 9
R33.0	31:0	Input Coverage Mask for Slot 8

DWord	Bits	Description
		R34-R57: Delivered only if the corresponding **Barycentric Interpolation Mode** bit is set and this is a *32-pixel dispatch*.
R34.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 23 This and the next register phase is only included if the corresponding enable bit in **Barycentric Interpolation Mode** is set. Format = IEEE_Float
R34.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 22
R34.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 21
R34.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 20
R34.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 19
R34.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 18
R34.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 17
R34.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 16
R35		Perspective Pixel Location Barycentric[2] for Slots 23:16
R36.7	31:0	Perspective Pixel Location Barycentric[1] for Slot 31
R36.6	31:0	Perspective Pixel Location Barycentric[1] for Slot 30
R36.5	31:0	Perspective Pixel Location Barycentric[1] for Slot 29
R36.4	31:0	Perspective Pixel Location Barycentric[1] for Slot 28
R36.3	31:0	Perspective Pixel Location Barycentric[1] for Slot 27
R36.2	31:0	Perspective Pixel Location Barycentric[1] for Slot 26
R36.1	31:0	Perspective Pixel Location Barycentric[1] for Slot 25
R36.0	31:0	Perspective Pixel Location Barycentric[1] for Slot 24
R37		Perspective Pixel Location Barycentric[2] for Slots 31:24
R38:41		Perspective Centroid Barycentric
R42:45		Perspective Sample Barycentric
R46:49		Linear Pixel Location Barycentric
R50:53		Linear Centroid Barycentric
R54:57		Linear Sample Barycentric
		R58-R59: Delivered only if **Pixel Shader Uses Source Depth** is set and this is a *32-pixel dispatch*.
R58.7	31:0	Interpolated Depth for Slot 23 Format = IEEE_Float This and the next register phase is only included if **Pixel Shader Uses Source Depth** (WM_STATE) bit is set.
R58.6	31:0	Interpolated Depth for Slot 22
R58.5	31:0	Interpolated Depth for Slot 21

DWord	Bits	Description
R58.4	31:0	Interpolated Depth for Slot 20
R58.3	31:0	Interpolated Depth for Slot 19
R58.2	31:0	Interpolated Depth for Slot 18
R58.1	31:0	Interpolated Depth for Slot 17
R58.0	31:0	Interpolated Depth for Slot 16
R59.7	31:0	Interpolated Depth for Slot 31
R59.6	31:0	Interpolated Depth for Slot 30
R59.5	31:0	Interpolated Depth for Slot 29
R59.4	31:0	Interpolated Depth for Slot 28
R59.3	31:0	Interpolated Depth for Slot 27
R59.2	31:0	Interpolated Depth for Slot 26
R59.1	31:0	Interpolated Depth for Slot 25
R59.0	31:0	Interpolated Depth for Slot 24
		R60-R61: Delivered only if **Pixel Shader Uses Source W** is set and this is a *32-pixel dispatch*.
R60.7	31:0	Interpolated W for Slot 23 Format = IEEE_Float This and the next register phase are only included if **Pixel Shader Uses Source W** (WM_STATE) bit is set.
R60.6	31:0	Interpolated W for Slot 22
R60.5	31:0	Interpolated W for Slot 21
R60.4	31:0	Interpolated W for Slot 20
R60.3	31:0	Interpolated W for Slot 19
R60.2	31:0	Interpolated W for Slot 18
R60.1	31:0	Interpolated W for Slot 17
R60.0	31:0	Interpolated W for Slot 16
R61.7	31:0	Interpolated W for Slot 31
R61.6	31:0	Interpolated W for Slot 30
R61.5	31:0	Interpolated W for Slot 29
R61.4	31:0	Interpolated W for Slot 28
R61.3	31:0	Interpolated W for Slot 27
R61.2	31:0	Interpolated W for Slot 26
R61.1	31:0	Interpolated W for Slot 25
R61.0	31:0	Interpolated W for Slot 24
		R62: Delivered only if **Position XY Offset Select** is either POSOFFSET_CENTROID or POSOFFSET_SAMPLE and this is a *32-pixel dispatch*.

DWord	Bits	Description
R62.7	31:24	Position Offset Y for Slot 31 This field contains either the CENTROID or SAMPLE position offset for Y, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	23:16	Position Offset X for Slot 31 This field contains either the CENTROID or SAMPLE position offset for X, depending on the state of **Position XY Offset Select**. Format = U4.4 Range = [0.0,1.0)
	15:8	Position Offset Y for Slot 30
	7:0	Position Offset X for Slot 30
R62.6	31:24	Position Offset Y for Slot 29
	23:16	Position Offset X for Slot 29
	15:8	Position Offset Y for Slot 28
	7:0	Position Offset X for Slot 28
R62.5:4		Position Offset X/Y for Slot[27:24]
R62.3:2		Position Offset X/Y for Slot[23:20]
R62.1:0		Position Offset X/Y for Slot[19:16]
		R63-R64: Delivered only if **Pixel Shader Uses Input Coverage Mask** is set and this is a *32-pixel dispatch*.
R63.7	31:0	Input Coverage Mask for Slot 23 Format = U32 This and the next register phase are only included if **Pixel Shader Uses Input Coverage Mask** (3DSTATE_PS) is set.
R63.6	31:0	Input Coverage Mask for Slot 22
R63.5	31:0	Input Coverage Mask for Slot 21
R63.4	31:0	Input Coverage Mask for Slot 20
R63.3	31:0	Input Coverage Mask for Slot 19
R63.2	31:0	Input Coverage Mask for Slot 18
R63.1	31:0	Input Coverage Mask for Slot 17
R63.0	31:0	Input Coverage Mask for Slot 16
R64.7	31:0	Input Coverage Mask for Slot 31
R64.6	31:0	Input Coverage Mask for Slot 30

DWord	Bits	Description
R64.5	31:0	Input Coverage Mask for Slot 29
R64.4	31:0	Input Coverage Mask for Slot 28
R64.3	31:0	Input Coverage Mask for Slot 27
R64.2	31:0	Input Coverage Mask for Slot 26
R64.1	31:0	Input Coverage Mask for Slot 25
R64.0	31:0	Input Coverage Mask for Slot 24

Pixel Backend

This section contains the following subsections:

- MCS Buffer for Render Target(s) [CHV, BSW]
- Render Target Fast Clear [CHV, BSW]
- Render TargetResolve [CHV, BSW]

Color Calculator (Output Merger)

Overview

Note: The Color Calculator logic resides in the Render Cache backing Data Port (DAP) shared function. It is described in this chapter as the Color Calc functions are naturally an extension of the 3D pipeline past the WM stage. See the DataPort chapter for details on the messages used by the Pixel Shader to invoke Color Calculator functionality.

The *Color Calculator* (referred to as "Output Merger in the DX Spec) function within the Data Port shared function completes the processing of rasterized pixels after the pixel color and depth have been computed by the Pixel Shader. This processing is initiated when the pixel shader thread sends a Render Target Write message (see *Shared Functions*) to the Render Cache. (Note that a single pixel shader thread may send multiple Render Target Write messages, with the result that multiple render targets get updated.) The pixel variables pass through a pipeline of fixed (yet programmable) functions, and the results are conditionally written into the appropriate buffers.

The word "pixel" used in this section is effectively replaced with the word "sample" if multisample rasterization is enabled.

Pipeline Stage	Description
Alpha Coverage	It generates coverage masks using AlphaToCoverage AND/OR AlphaToOne functions based on src0.alpha.
Alpha Test	Compare pixel alpha with reference alpha and conditionally discard pixel.
Stencil Test	Compare pixel stencil value with reference and forward result to Buffer Update stage.
Depth Test	Compare pix.Z with corresponding Z value in the Depth Buffer and forward result to Buffer Update stage.
Color Blending	Combine pixel color with corresponding color in color buffer according to programmable function.
Gamma Correction	Adjust pixel's color according to gamma function for SRGB destination surfaces.
Color Quantization	Convert "full precision" pixel color values to fixed precision of the color buffer format.
Logic Ops	Combine pixel color logically with existing color buffer color (mutually exclusive with Color Blending).
Buffer Update	Write final pixel values to color and depth buffers or discard pixel without update.

The following logic describes the high-level operation of the Pixel Processing pipeline:

```
PixelProcessing() {
    AlphaCoverage()
    AlphaTest()
    DepthBufferCoordinateOffsetDisable
    StencilTest()
    DepthTest()
    ColorBufferBlending()
    GammaCorrection()
    ColorQuantization()
    LogicalOps()
    BufferUpdate()
}
```

Alpha Coverage

Alpha coverage logic is supported for CHV, BSW and can be controlled using three state variables:

- **AlphaToCoverage Enable**, when enabled Color Calculator modifies the sample mask. This function (along with AlphaToOne) come at the top of the pixel pipeline. The sample's Source0.Alpha value (possibly being replicated from the pixel's Source0.Alpha) is used to compute a (optionally dithered) 1/2/4-bit mask (depending on NumSamples).
- The **AlphaToCoverage Dither Enable** SV is used to control the dithering of the AlphaToCoverage mask. The bit corresponding to the sample# is then ANDed with the sample's incoming mask bits – allowing the sample to be masked off depending on alpha.
- **AlphaToOne Enable,** when enabled, Color Calculator must replace Source0.Alpha (if present) with 1.0f.
- If AlphaToCoverage is disabled, AlphaToCoverage Dither does not have any impact.

If Pixel Shader outputs oMask, AlphaToCoverage is disabled in hardware, regardless of the state setting for this feature.

Notes:

- Src0.alpha needs to be first multiplied with AA alpha before applying AlphaToCoverage and AlphaToOne functions.
- An alpha value of NaN results in a no coverage (zero) mask.
- Alpha values from the pixel shader are treated as FLOAT32 format for computing the AlphaToCoverage Mask.

Alpha Test

The Alpha Test function can be used to discard pixels based on a comparison between the incoming pixel's alpha value and the **Alpha Test Reference** state variable in COLOR_CALC_STATE. This operation can be used to remove transparent or nearly-transparent pixels, though other uses for the alpha channel and alpha test are certainly possible.

This function is enabled by the **Alpha Test Enable** state variable in COLOR_CALC_STATE. If ENABLED, this function compares the incoming pixel's alpha value (*pixColor.Alpha*) and the reference alpha value

specified by via the **Alpha Test Reference** state variable in COLOR_CALC_STATE. The comparison performed is specified by the **Alpha Test Function** state variable in COLOR_CALC_STATE.

The **Alpha Test Format** state variable is used to specify whether Alpha Test is performed using fixed-point (UNORM8) or FLOAT32 values. Accordingly, it determines whether the **Alpha Reference Value** is passed in a UNORM8 or FLOAT32 format. If UNORM8 is selected, the pixel's alpha value will be converted from floating-point to UNORM8 before the comparison.

Pixels that pass the Alpha Test proceed for further processing. Those that fail are discarded at this point in the pipeline.

If **Alpha Test Enable** is DISABLED, this pipeline stage has no effect.

The Alpha Test function is supported in conjunction with Multiple Render Targets (MRTs). If delivered in the incoming render target write message, source 0 alpha is used to perform the alpha test. If source 0 alpha is not delivered, the normal alpha value is used to perform the alpha test.

Depth Coordinate Offset

The Depth Coordinate Offset function applies a programmable constant offset to the RenderTarget X,Y screen space coordinates in order to generate DepthBuffer coordinates.

The function has been specifically added to allow the OpenGL driver to deal with a RenderTarget and DepthBuffer of differing sizes.

OpenGL defines a lower-left screen coordinate origin. This requires the driver to incorporate a "Y coordinate flipping" transformation into the viewport mapping function. The Y extent of the RT is used in this flipping transformation. If the DepthBuffer extent is different, the wrong pixel Y locations within the DepthBuffer will be accessed.

The least expensive solution is to provide a translation offset to be applied to the post-viewport-mapped DepthBuffer Y pixel coordinate, effectively allowing the alignment of the lower-left origins of the RT and DepthBuffer. [Note that the previous DBCOD feature performed an optional translation of post-viewport-mapping RT pixel (screen) coordinates to generate DepthBuffer pixel (window) coordinates. Specifically, the Draw Rect Origin X,Y state could be subtracted from the RT pixel coordinates.]

This function uses **Depth Coordinate Offset X,Y** state (signed 16-bit values in 3DSTATE_DEPTH_RECTANGLE) that is unconditionally added to the RT pixel coordinates to generate DepthBuffer pixel coordinates.

The previous DBCOB feature can be supported by having the driver program Depth Coordinate X,Y Offset to the two's complement of the the Draw Rect Origin. By programming Depth Coordinate X,Y Offset to zeros, the current "normal" operation (DBCOD disabled) can be achieved.

Programming Note	
Context:	Depth Coordinate Offset
• Only simple 2D RTs are supported (no mipmaps). • Software must ensure that the resultant DepthBuffer Coordinate X,Y values are non-negative. • There are alignment restrictions – see 3DSTATE_DEPTH_BUFFER command.on SFID_DP_DC2) are IA coherent.	

Stencil Test

The Stencil Test function can be used to discard pixels based on a comparison between the [**Backface**] **Stencil Test Reference** state variable and the pixel's stencil value. This is a general purpose function used for such effects as shadow volumes, per-pixel clipping, etc. The result of this comparison is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Stencil Test Enable** state variable. If ENABLED, the current stencil buffer value for this pixel is read.

A 2nd set of the stencil test state variables is provided so that pixels from back-facing objects, assuming they are not culled, can have a stencil test performed on them separate from the test for normal front-facing objects. The separate stencil test for back-facing objects can be enabled via the **Double Sided Stencil Enable** state variable. Otherwise, non-culled back-facing objects will use the same test function, mask and reference value as front-facing objects. The 2nd stencil state for back-facing objects is most commonly used to improve the performance of rendering shadow volumes which require a different stencil buffer operation depending on whether pixels rendered are from a front-facing or back-facing object. The backface stencil state removes the requirement to render the shadow volumes in 2 passes or sort the objects into front-facing and back-facing lists.

The remainder of this subsection describes the function in term of [**Backface**] <**state variable name**>. The Backface set of state variables are only used if Double Sided Stencil Enable is ENABLED and the object is considered back-facing. Otherwise the normal (front-facing) state variables are used.

This function then compares the [**Backface**] **Stencil Test Reference** value and the pixel's stencil value value after logically ANDing both values by [**Backface**] **Stencil Test Mask**. The comparison performed is specified by the [**Backface**] **Stencil Test Function** state variable. The result of the comparison is passed down the pipeline for use in the Stencil Buffer Update function. The Stencil Test function does not in itself discard pixels.

If **Stencil Test Enable** is DISABLED, a result of "stencil test passed" is propagated down the pipeline.

Depth Test

The Depth Test function can be used to discard pixels based on a comparison between the incoming pixel's depth value and the current depth buffer value associated with the pixel. This function is typically used to perform the "Z Buffer" hidden surface removal. The result of this pipeline function is used in the Stencil Buffer Update function later in the pipeline.

This function is enabled by the **Depth Test Enable** state variable. If enabled, the pixel's ("source") depth value is first computed. After computation the pixel's depth value is clamped to the range defined by **Minimum Depth** and **Maximum Depth** in the selected CC_VIEWPORT state. Then the current ("destination") depth buffer value for this pixel is read.

This function then compares the source and destination depth values. The comparison performed is specified by the **Depth Test Function** state variable.

The result of the comparison is propogated down the pipeline for use in the subsequent Depth Buffer Update function. The Depth Test function does not in itself discard pixels.

If **Depth Test Enable** is DISABLED, a result of "depth test passed" is propagated down the pipeline.

Programming Note:

- Enabling the Depth Test function without defining a Depth Buffer is UNDEFINED.

Pre-Blend Color Clamping

Pre-Blend Color Clamping, controlled via **Pre-Blend Color Clamp Enable** OR **Pre-Blend Source Only Clamp Enable** and **Color Clamp Range** states in COLOR_CALC_STATE, is affected by the enabling of Color Buffer Blend as described below.

The following table summarizes the requirements involved with Pre-/Post-Blend Color Clamping.

Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
Off	UNORM,	Must be enabled with range = RT range or [0,1] (same function)	N/A, state ignored
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	N/A, state ignored
	FLOAT (except for R11G11B10_FLOAT)	Must be enabled (with any desired range)	N/A, state ignored
	R11G11B10_FLOAT	Must be enabled with either [0,1] or RT range	N/A, state ignored
	UINT, SINT	State ignored, implied clamp to RT range	N/A, state ignored
On (where permitted)	UNORM, UNORM_SRGB	Must be enabled with range = RT range or [0,1] (same function)	Must be enabled with range = RT range or [0,1] (same function)

Blending	RT Format	Pre-Blend Color Clamp	Post-Blend Color Clamp
	SNORM	Must be enabled with range = RT range or [-1,1] (same function)	Must be enabled with range = RT range or [-1,1] (same function)
	FLOAT (except for R11G11B10_FLOAT)	Can be disabled or enabled (with any desired range)	Must be enabled (with any desired range)
	R11G11B10_FLOAT	Can be disabled or enabled (with any desired range)	Must be enabled with either [0,1] or RT range

Pre-Blend Color Clamping When Blending is Disabled

The clamping of source color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all source color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed.

Programming Note	
Context:	Pre-Blend Color Clamping When Blending is Disabled

- Given the possibility of writing UNPREDICTABLE values to the Color Buffer, it is expected and highly recommended that, when blending is disabled, software set **Pre-Blend Color Clamp Enable** to ENABLED and select an appropriate **Color Clamp Range**.
- When using SINT or UINT rendertarget surface formats, **Blending must** be DISABLED. The **Pre-Blend Color Clamp Enable** and **Color Clamp Range** fields are ignored, and an implied clamp to the rendertarget surface format is performed.

Pre-Blend Color Clamping When Blending is Enabled

The clamping of source, destination and constant color components is controlled by **Pre-Blend Color Clamp Enable**. If ENABLED, all these color components are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed on these color components prior to blending.

Color Buffer Blending

The Color Buffer Blending function is used to combine one or two incoming "source" pixel color+alpha values with the "destination" color+alpha read from the corresponding location in a RenderTarget.

Blending is enabled on a global basis by the **Color Buffer Blend Enable** state variable (in COLOR_CALC_STATE). If DISABLED, Blending and Post-Blend Clamp functions are disabled for all RenderTargets, and the pixel values (possibly subject to Pre-Blend Clamp) are passed through unchanged.

The Color Buffer Blend Enable is in the per-render-target BLEND_STATE, and the field in SURFACE_STATE is no longer supported.

Programming Note	
Context:	Color Buffer Blending, Logic Ops, DataPort, surface formats, render targets

- Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
- Dual source blending: The DataPort only supports dual source blending with a SIMD8-style message.
- Only certain surface formats support Color Buffer Blending. Refer to the Surface Format tables in *Sampling Engine*. Blending must be disabled on a RenderTarget if blending is not supported.

The incoming "source" pixel values are modulated by a selected "source" blend factor, and the possibly gamma-decorrected "destination" values are modulated by a "destination" blend factor. These terms are then combined with a "blend function". In general:

src_term = src_blend_factor * src_color

dst_term = dst_blend_factor * dst_color

color output = blend_function(src_term, dst_term)

If there is no alpha value contained in the Color Buffer, a default value of 1.0 is used and, correspondingly, there is no alpha component computed by this function.

Dual Source Blending: When using "Dual Source" Render Target Write messages, the Source1 pixel color+alpha passed in the message can be selected as a src/dst blend factor. See Color Buffer Blend Color Factors. In single-source mode, those blend factor selections are invalid. If SRC1 is included in a src/dst blend factor and a DualSource RT Write message is not used, results are UNDEFINED. (This reflects the same restriction in DX APIs, where undefined results are produced if "o1" is not written by a PS – there are no default values defined). If SRC1 is not included in a src/dst blend factor, dual source blending must be disabled.

The blending of the color and alpha components is controlled with two separate (color and alpha) sets of state variables. However, if the **Independent Alpha Blend Enable** state variable in COLOR_CALC_STATE is DISABLED, then the "color" (rather than "alpha") set of state variables is used for both color and alpha. Note that this is the only use of the **Independent Alpha Blend Enable** state – it does not control whether Blending occurs, only how.

Per **Render Target Blend State**: Blend state is selected based on **Render Target Index** contained in the message header, and appropriate blend state is applied to Render Target Write messages.

The following table describes the color source and destination blend factors controlled by the **Source [Alpha] Blend Factor** and **Destination [Alpha] Blend Factor** state variables in COLOR_CALC_STATE. Note that the blend factors applied to the R,G,B channels are always controlled by the **Source/Destination Blend Factor**, while the blend factor applied to the alpha channel is controlled either by **Source/Destination Blend Factor** or **Source/Destination Alpha Blend Factor**.

Color Buffer Blend Color Factors

Blend Factor Selection	Blend Factor Applied for R,G,B,A channels (oN = output from PS to RT#N) (o1 = 2nd output from PS in Dual-Souce mode only) (rtN = destination color from RT#N) (CC = Constant Color)
BLENDFACTOR_ZERO	0.0, 0.0, 0.0, 0.0
BLENDFACTOR_ONE	1.0, 1.0, 1.0, 1.0
BLENDFACTOR_SRC_COLOR	oN.r, oN.g, oN.b, oN.a
BLENDFACTOR_INV_SRC_COLOR	1.0-oN.r, 1.0-oN.g, 1.0-oN.b, 1.0-oN.a
BLENDFACTOR_SRC_ALPHA	oN.a, oN.a, oN.a, oN.a
BLENDFACTOR_INV_SRC_ALPHA	1.0-oN.a, 1.0-oN.a, 1.0-oN.a, 1.0-oN.a
BLENDFACTOR_SRC1_COLOR	o1.r, o1.g, o1.b, o1.a
BLENDFACTOR_INV_SRC1_COLOR	1.0-o1.r, 1.0-o1.g, 1.0-o1.b, 1.0-o1.a
BLENDFACTOR_SRC1_ALPHA	o1.a, o1.a, o1.a, o1.a
BLENDFACTOR_INV_SRC1_ALPHA	1.0-o1.a, 1.0-o1.a, 1.0-o1.a, 1.0-o1.a
BLENDFACTOR_DST_COLOR	rtN.r, rtN.g, rtN.b, rtN.a
BLENDFACTOR_INV_DST_COLOR	1.0-rtN.r, 1.0-rtN.g, 1.0-rtN.b, 1.0-rtN.a
BLENDFACTOR_DST_ALPHA	rtN.a, rtN.a, rtN.a, rtN.a
BLENDFACTOR_INV_DST_ALPHA	1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a, 1.0-rtN.a
BLENDFACTOR_CONST_COLOR	CC.r, CC.g, CC.b, CC.a
BLENDFACTOR_INV_CONST_COLOR	1.0-CC.r, 1.0-CC.g, 1.0-CC.b, 1.0-CC.a
BLENDFACTOR_CONST_ALPHA	CC.a, CC.a, CC.a, CC.a
BLENDFACTOR_INV_CONST_ALPHA	1.0-CC.a, 1.0-CC.a, 1.0-CC.a, 1.0-CC.a
BLENDFACTOR_SRC_ALPHA_SATURATE	f,f,f,1.0 where f = min(1.0 – rtN.a, oN.a)

The following table lists the supported blending operations defined by the **Color Blend Function** state variable and the **Alpha Blend Function** state variable (when in independent alpha blend mode).

Color Buffer Blend Functions

Blend Function	Operation (for each color component)
BLENDFUNCTION_ADD	SrcColor*SrcFactor + DstColor*DstFactor
BLENDFUNCTION_SUBTRACT	SrcColor*SrcFactor - DstColor*DstFactor
BLENDFUNCTION_REVERSE_SUBTRACT	DstColor*DstFactor - SrcColor*SrcFactor

Blend Function	Operation (for each color component)
BLENDFUNCTION_MIN	min (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "min" function.
BLENDFUNCTION_MAX	max (SrcColor*SrcFactor, DstColor*DstFactor) **Programming Note:** This is a superset of the OpenGL "max" function.

Post-Blend Color Clamping

(See *Pre-Blend Color Clamping* above for a summary table regarding clamping)

Post-Blend Color clamping is available only if Blending is enabled.

If Blending is enabled, the clamping of blending output color components is controlled by **Post-Blend Color Clamp Enable**. If ENABLED, the color components output from blending are clamped to the range specified by **Color Clamp Range**. If DISABLED, no clamping is performed at this point.

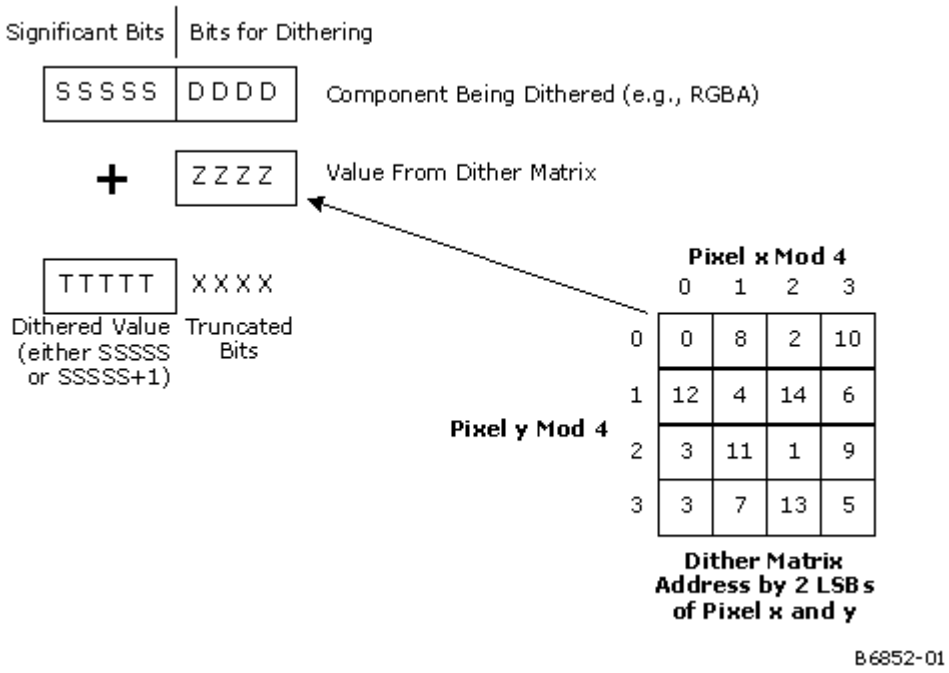
Regardless of the setting of **Post-Blend Color Clamp Enable**, when Blending is enabled color components will be automatically clamped to (at least) the rendertarget surface format range at this stage of the pipeline.

Dithering

Dithering is used to give the illusion of a higher resolution when using low-bpp channels in color buffers (e.g., with 16bpp color buffer). By carefully choosing an arrangement of lower resolution colors, colors otherwise not representable can be approximated, especially when seen at a distance where the viewer's eyes will average adjacent pixel colors. Color dithering tends to diffuse the sharp color bands seen on smooth-shaded objects.

A four-bit dither value is obtained from a 4x4 Dither Constant matrix depending on the pixel's X and Y screen coordinate. The pixel's X and Y screen coordinates are first offset by the **Dither Offset X** and **Dither Offset Y** state variables (these offsets are used to provide window-relative dithering). Then the two LSBs of the pixel's screen X coordinate are used to address a column in the dither matrix, and the two LSBs of the pixel's screen Y coordinate are used to address a row. This way, the matrix repeats every four pixels in both directions.

The value obtained is appropriately shifted to align with (what would be otherwise) truncated bits of the component being dithered. It is then added with the component and the result is truncated to the bit depth of the component given the color buffer format.



Dithering Process (5-Bit Example)

Logic Ops

The Logic Ops function is used to combine the incoming "source" pixel color/alpha values with the corresponding "destination" color/alpha contained in the ColorBuffer, using a logic function.

The Logic Op function is enabled by the **LogicOp Enable** state variable. If DISABLED, this function is ignored and the incoming pixel values are passed through unchanged.

Programming Notes

Programming Note
Color Buffer Blending and Logic Ops must not be enabled simultaneously, or behavior is UNDEFINED.
Logic Ops are supported on all blendable render targets and render targets with *INT formats.

The following table lists the supported logic ops. The logic op is selected using the **Logic Op Function** field in COLOR_CALC_STATE.

Logic Ops

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_CLEAR	all 0's
LOGICOP_NOR	NOT (S OR D)
LOGICOP_AND_INVERTED	(NOT S) AND D
LOGICOP_COPY_INVERTED	NOT S
LOGICOP_AND_REVERSE	S AND NOT D

LogicOp Function	Definition (S=Source, D=Destination)
LOGICOP_INVERT	NOT D
LOGICOP_XOR	S XOR D
LOGICOP_NAND	NOT (S AND D)
LOGICOP_AND	S AND D
LOGICOP_EQUIV	NOT (S XOR D)
LOGICOP_NOOP	D
LOGICOP_OR_INVERTED	(NOT S) OR D
LOGICOP_COPY	S
LOGICOP_OR_REVERSE	S OR NOT D
LOGICOP_OR	S OR D
LOGICOP_SET	all 1's

Buffer Update

The Buffer Update function is responsible for updating the pixel's Stencil, Depth and Color Buffer contents based upon the results of the Stencil and Depth Test functions. Note that Kill Pixel and/or Alpha Test functions may have already discarded the pixel by this point.

Stencil Buffer Updates

If and only if stencil testing is enabled, the Stencil Buffer is updated according to the **Stencil Fail Op**, **Stencil Pass Depth Fail Op**, and **Stencil Pass Depth Pass Op** state (or their backface counterparts if **Double Sided Stencil Enable** is ENABLED and the pixel is from a back-facing object) and the results of the Stencil Test and Depth Test functions.

Stencil Fail Op and **Backface Stencil Fail Op** specify how/if the stencil buffer is modified if the stencil test fails. **Stencil Pass Depth Fail Op** and **Backface Stencil Pass Depth Fail Op** specify how/if the stencil buffer is modified if the stencil test passes but the depth test fails. **Stencil Pass Depth Pass Op** and **Backface Stencil Pass Depth Pass Op** specify how/if the stencil buffer is modified if both the stencil and depth tests pass. The operations (on the stencil buffer) that are to be performed under one of these (mutually exclusive) conditions is summarized in the following table.

Stencil Buffer Operations

Stencil Operation	Description
STENCILOP_KEEP	Do not modify the stencil buffer
STENCILOP_ZERO	Store a 0
STENCILOP_REPLACE	Store the *StencilTestReference* reference value
STENCILOP_INCRSAT	Saturating increment (clamp to max value)
STENCILOP_DECRSAT	Saturating decrement (clamp to 0)
STENCILOP_INCR	Increment (possible wrap around to 0)

Stencil Operation	Description
STENCILOP_DECR	Decrement (possible wrap to max value)
STENCILOP_INVERT	Logically invert the stencil value

Any and all writes to the stencil portion of the depth buffer are enabled by the **Stencil Buffer Write Enable** state variable.

When writes are enabled, the **Stencil Buffer Write Mask** and **Backface Stencil Buffer Write Mask** state variables provide an 8-bit mask that selects which bits of the stencil write value are modified. Masked-off bits (i.e., mask bit == 0) are left unmodified in the Stencil Buffer.

Programming Note	
Context:	Stencil Buffer Updates
The Stencil Buffer can be written even if depth buffer writes are disabled via Depth Buffer Write Enable	

Depth Buffer Updates

Any and all writes to the Depth Buffer are enabled by the **Depth Buffer Write Enable** state variable. If there is no Depth Buffer, writes must be explicitly disabled with this state variable, or operation is UNDEFINED.

If depth testing is disabled or the depth test passed, the incoming pixel's depth value is written to the Depth Buffer. If depth testing is enabled and the depth test failed, the pixel is discarded – with no modification to the Depth or Color Buffers (though the Stencil Buffer may have been modified).

Color Gamma Correction

Computed RGB (not A) channels can be gamma-corrected prior to update of the Color Buffer.

This function is automatically invoked whenever the destination surface (render target) has an SRGB format (see surface formats in *Sampling Engine*). For these surfaces, the computed RGB values are converted from gamma=1.0 space to gamma=2.4 space by applying a ^(2.4) exponential function.

Color Buffer Updates

Finally, if the pixel has not been discarded by this point, the incoming pixel color is written into the Color Buffer. The **Surface Format** of the color buffer indicates which channel(s) are written (e.g., R8G8_UNORM are written with the Red and Green channels only). The **Color Buffer Component Write Disables** from the Color Buffer's SURFACE_STATE provide an independent write disable for each channel of the Color Buffer.

Pixel Pipeline State Summary

COLOR_CALC_STATE

This topic is currently under development.

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_BLEND_STATE_POINTERS

3DSTATE_DEPTH_STENCIL_STATE_POINTERS

3DSTATE_DEPTH_STENCIL_STATE_POINTERS does not exist. It has been replaced by 3DSTATE_WM_DEPTH_STENCIL. (See Vol2a.11 3D Pipeline – Windower for details.)

COLOR_CALC_STATE

COLOR_CALC_STATE

DEPTH_STENCIL_STATE

DEPTH_STENCIL_STATE does not exist. It has been replaced by 3DSTATE_WM_DEPTH_STENCIL. (See *3D Pipeline – Windower* for details).

BLEND_STATE

BLEND_STATE

Signal [CHV, BSW]	CC_INT::AlphaTestEnable
Description	AlphaTestEnable
Formula	= BLEND_STATE::AlphaTestEnable && !3DSTATE_WM_HZ_OP::DepthBufferResolveEnable && !3DSTATE_WM_HZ_OP::DepthBufferClear && !3DSTATE_WM_HZ_OP::StencilBufferClear

Signal [CHV, BSW]	CC_INT::AlphaToCoverageEnable
Description	AlphaToCoverageEnable
Formula	= BLEND_STATE::AlphaToCoverageEnable && !3DSTATE_PS_EXTRA::PixelShaderDisableAlphaToCoverage

CC_VIEWPORT

CC_VIEWPORT

Other Pixel Pipeline Functions

Statistics Gathering

If **Statistics Enable** is set in 3DSTATE_WM, the PS_DEPTH_COUNT register (see Memory Interface Registers in Volume 1a, *GPU Overview*) is incremented once for each pixel (or sample) that passes the depth, stencil and alpha tests. Note that each of these tests is treated as passing if disabled. This count is accurate regardless of whether **Early Depth Test Enable** is set. To obtain the value from this register at a deterministic place in the primitive stream without flushing the pipeline, however, the PIPE_CONTROL command must be used. See Volume 2a, *3D Pipeline*, for details on PIPE_CONTROL.

MCS Buffer for Render Target(s)

MCS buffer can be enabled for two purposes described below. MCS buffer can be controlled using two mechanisms:

1. MMIO bit Cache Mode 1 (0x2124) register bit 5
2. RT surface state

The following table summarizes modes of operation related to MCS buffer:

Cache Mode MMIO Bit (Please refer to Vol 1c)	MSC Enable (Surface State)	Operation
1 (feature disable)	X	Normal mode of operation i.e. no MSAA compression and no color clear
0	0	Normal mode of operation i.e. no MSAA compression and no color clear
0	1	Depending on the Number of multi-samples, either MSAA compression OR color clear is enabled

Project	MSAA	Width of Clear Rect	Height of Clear Rect
CHV, BSW	2x	Ceil(1/8*width)	Ceil(1/2*height)
CHV, BSW	4X	Ceil(1/8*width)	Ceil(1/2*height)
CHV, BSW	8X	Ceil(1/2*width)	Ceil(1/2*height)

- **MSAA Compression:** Multi-sample render target is bound to the pipeline and MSAA compression feature is enabled. In this case, MCS buffer stores the information required for MSAA compression algorithm. The size and layout of the MCS buffer is based on per-pixel RT. For 4X and 8X MSAA, MCS buffer element is 8bpp and 32bpp respectively. Height, width, and layout of MCS buffer in this case must match with Render Target height, width, and layout. MCS buffer is tiledY. When MCS buffer is enabled and bound to MSRT, it is required that it is cleared prior to any rendering. A clear value can be specified optionally in the surface state of the corresponding RT. Clear pass for this case requires that scaled down primitive is sent down with upper left

coordinate to coincide with actual rectangle being cleared. For MSAA, clear rectangle's height and width need to as show in the following table in terms of (width,height) of the RT.

- **Fast Color Clear:** When non multi-sample render target is bond to the pipeline and MSC buffer is enabled, MCS buffer is used as an intermediate (coarse granular) buffer per RT. Hence, MCS buffer is used to improve render target clear. When MCS is buffer is used for color clear of non-multisampler render target, the following restrictions apply:

Color Clear of Non-MultiSampler Render Target Restrictions

<table>
<tr><th>Restrictions</th></tr>
<tr><td>Support is limited to tiled render targets.</td></tr>
<tr><td>Mip-mapped and arrayed surfaces are supported with MCS buffer layout with these alignments in the RT space: Horizontal Alignment = 256 and Vertical Alignment = 128.</td></tr>
<tr><td>Clear is supported only on the full RT; i.e., no partial clear or overlapping clears.</td></tr>
<tr><td>The following table describes the RT alignment:
<table>
<tr><th></th><th>Pixels</th><th>Lines</th></tr>
<tr><th>TiledY RT CL</th><th></th><th></th></tr>
<tr><td>bpp</td><td></td><td></td></tr>
<tr><td>32</td><td>8</td><td>4</td></tr>
<tr><td>64</td><td>4</td><td>4</td></tr>
<tr><td>128</td><td>2</td><td>4</td></tr>
<tr><th>TiledX RT CL</th><th></th><th></th></tr>
<tr><td>bpp</td><td></td><td></td></tr>
<tr><td>32</td><td>16</td><td>2</td></tr>
<tr><td>64</td><td>8</td><td>2</td></tr>
<tr><td>128</td><td>4</td><td>2</td></tr>
</table>
</td></tr>
<tr><td>MCS buffer for non-MSRT is supported only for RT formats 32bpp, 64bpp, and 128bpp.</td></tr>
<tr><td>Clear pass must have a clear rectangle that must follow alignment rules in terms of pixels and lines as shown in the table below. Further, the clear-rectangle height and width must be multiple of the following dimensions. If the height and width of the render target being cleared do not meet these requirements, an MCS buffer can be created such that it follows the requirement and covers the RT.</td></tr>
<tr><td>Clear rectangle must be aligned to two times the number of pixels in the table shown below due to 16X16 hashing across the slice.</td></tr>
</table>

	Pixels	Lines
TiledY RT		
bpp		
32	128	128
64	64	128
128	32	128

TiledX RT		
bpp		
32	256	64
64	128	64
128	64	64

To optimize the performance MCS buffer (when bound to 1X RT) clear similarly to MCS buffer clear for MSRT case, clear rect is required to be scaled by the following factors in the horizontal and vertical directions:

	Horizontal Scale Down Factor	Vertical Scale Down Factor
MCS CL for TiledY RCC		
bpp		
32	64	64
64	32	64
128	16	64
MCS CL for TiledX RCC		
bpp		
32	128	32
64	64	32
128	32	32

Resolve rectangle must not be scaled if MCS Resolve Optimization is disabled in the Cache Mode register.

The following SW requirements for MCS buffer clear functionality apply in addition to the general SW requirements listed below:

- For non-MSRTs, loss less compression of render targets is supported for 32, 64, and 128 bpp surfaces as described in the Shared Functions Data Port under Render Target Write. Lossless RT compression can be enabled for each RT surface with auxiliary surface control bits as described in the Surface State. This feature changes the MSC layout for various tiled and BPP formats as described in above sections. Fast clear, render, and resolve operations are fundamentally the same. Since Sampler support reading this auxiliary (MCS) buffer for non-MSRTs, resolve passes can be avoided in the cases when fast cleared and possibly compressed RTs are consumed by the sampler.
- SW does not need to compile any PS for clear and resolve passes but must ensure that PS dispatch enable bit is set.
- Any transition from any value in {Clear, Render, Resolve} to a different value in {Clear, Render, Resolve} requires end of pipe synchronization.

The following are the general SW requirements for MCS buffer clear functionality:

- At the time of Render Target creation, SW needs to create clear-buffer, i.e., MCS buffer.

- At the clear time, clear value for that RT must be programmed and clear enable bit must be set in the surface state of the corresponding RT.
- SW must clear the RT with setting a RT clear bit set in the PS state during the clear pass as described in the following sub-section.
- Since only one RT is bound with a clear pass, only one RT can be cleared at a time. To clear multiple RTs, multiple clear passes are required.
- If Software wants to enable Color Compression without Fast clear, Software needs to initialize MCS with zeros.
- Lossless compression and CCS initialized to all F (using HW Fast Clear or SW direct Clear) on the same surface is not supported
- Before binding the "cleared" RT to texture OR honoring a CPU lock OR submitting for flip, SW must ensure a resolve pass. Such a resolve pass is described in the following sub-section.

Render Target Fast Clear

Fast clear of the render target is performed by setting the **Render Target Fast Clear Enable** field in 3DSTATE_PS and rendering a rectangle The size of the rectangle is related to the size of the MCS.

The following is required when performing a render target fast clear:

- The render target(s) is/are bound as they normally would be, with the MCS surface defined in SURFACE_STATE.
- A rectangle primitive of the same size as the MCS surface is delivered.
- The pixel shader kernel requires no attributes, and delivers a value of 0xFFFFFFFF in all channels of the render target write message The replicated color message should be used.
- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable,** and **Alpha Test Enable** must all be disabled.
- After Render target fast clear, pipe-control with color cache write-flush must be issued before sending any DRAW commands on that render target.

Render Target Resolve

If the MCS is enabled on a non-multisampled render target, the render target must be resolved before being used for other purposes (display, texture, CPU lock) The clear value from SURFACE_STATE is written into pixels in the render target indicated as clear in the MCS. This is done by setting the **Render Target Resolve Enable** field in 3DSTATE_PS and rendering a full render target sized rectangle. Once this is complete, the render target will contain the same contents as it would have had the rendering been performed with MCS surface disabled. In a typical usage model, the render target(s) need to be resolved after rendering and before using it as a source for any consecutive operation.

When performing a render target resolve, PIPE_CONTROL with end of pipe sync must be delivered.

A rectangle primitive must be scaled down by the following factors with respect to render target being resolved:

	Horizontal scale down factor	Vertical scale down factor
MSC CL for TiledY RCC		
bpp		
32	64	64
64	32	64
128	16	64
MSC CL for TiledX RCC		
bpp		
32	128	32
64	64	32
128	32	32

The pixel shader kernel requires no attributes, but must deliver a render target write message covering all pixels and all render targets desired to be resolved The color data in these messages is ignored (the replicated color message is required).

Programming Note	
Context:	Render Target Resolve

- **Depth Test Enable, Depth Buffer Write Enable, Stencil Test Enable, Stencil Buffer Write Enable,** and **Alpha Test Enable** must all be disabled.
- This render target resolve procedure is not supported on multisampled render targets. Unresolved multisampled render targets are directly supported by the sampling engine, which resolves clear values in addition to decompressing the surface This applies to both *ld2dms* and *sample2dms* messages.

Media GPGPU Pipeline

This section of the BSpec discusses Programming the GPGPU Pipeline, Thread Group Tracking, Generic Media, and other related topics.

Media GPGPU Pipeline

This section of the BSpec discusses Programming the GPGPU Pipeline, Thread Group Tracking, Generic Media, and other related topics.

Programming the GPGPU Pipeline

1. In MEDIA_VFE_STATE choose whether to set DW2.6 Bypass Gateway Control. Usually this will be set, allowing the gateway to be used without OpenGateway/CloseGateway.
2. Set up interface descriptor with # of threads in barrier. The barrier id is not specified here because can Gen7 automatically assigns barriers to thread groups when they are free. The amount of CURBE data to deliver per thread dispatch is set in the interface descriptor.
3. Set up CURBE with thread ids and common data for all thread dispatches in the thread group.
4. Set up a GPGPU_WALKER command or a set of GPGPU_OBJECT commands with the thread group ids to dispatch the threads. The CURBE data is sent in sections for each thread dispatch in the thread group; a new thread group starts sending the CURBE data from the beginning of the buffer.

Note: Gen7 can either have the barriers and SLM automatically managed by hardware or specified by software. Mixing software managed and hardware managed in the same set of threads is allowed, but may cause stalls if there is an allocation conflict.

Note: When using GPGPU_OBJECT, finish dispatching a thread group before starting a different one.

The kernel should handle the barriers as follows:

The BarrierMsg message contains the barrier id and a way to reprogram the barrier count. The barrier count reprogram is not normally used for GPGPU workloads. When all threads in the group have reached the barrier, the gateway returns a notification bit 0.

The kernel must wait for the barrier to finish with a WAIT N0.

If a GPGPU kernel is going to access the Render Cache then the Fixed Function DOP Clock Gate Disable should be set to prevent the clock gating of the DAP used to access the Render Cache. This should be reset as soon as all the kernels that are accessing the Render Cache are finished and the Render cache flushed.

GPGPU Thread Limits

GPGPU requires 1024 SIMD channels to be available for a maximum size thread group. In a system with 10 EUs per subslice, each with 7 hardware threads, this means that a maximum size thread group will fit in a subslice if SIMD16 instructions are used, but not if SIMD8 are used. These limits can be circumvented for thread groups which do not need access to a barrier or SLM, in which case the thread group can cross sub-slices.

The Configurations section should be referenced to determine what SIMD is required to fit in the subslice of the targeted configuration.

GPGPU Commands

This section contains various commands for GPGPU, including a number of them shared with media mode.

MEDIA_VFE_STATE with varying definitions for different generations and projects:

MEDIA_VFE_STATE

MEDIA_CURBE_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Interface Descriptor Data payload as pointed to by the Interface Descriptor Data Start Address, with varying definitions for different generations and projects:

INTERFACE_DESCRIPTOR_DATA

MEDIA_STATE_FLUSH

GPGPU_WALKER

GPGPU Command Workarounds

GPGPU Commands have some subtle programming restrictions and workarounds to be aware of, as described below.

The MEDIA_STATE_FLUSH command is updated to specify all the resources required for the next thread group via an interface descriptor – if the resources are not available the group cannot start.

GPGPU Indirect Thread Dispatch

Indirect thread dispatch allows one thread group to control the group size of a following thread group.

This is the sequence of commands in the ring buffer:

GPGPU_OBJECT/WALKER	// Either a set of objects or a walker to dispatch a thread group which will write the next group's properties to memory.
MI_FLUSH	// Make sure the thread group has finished executing.
MEDIA_CURBE_LOAD	// Load the thread ids for new group.
MI_LOAD_REGISTER_MEMORY	// Load the indirect MMIO GPGPU registers from the mem written by the previous group.
GPGPU_WALKER (indirect)	// A walker with the indirect bit set.

The first thread group writes this data to memory:

1. The thread ids delivered in the CURBE - written where the following MEDIA_CURBE_LOAD will read them.
2. The GPGPU_WALKER parameters are written to memory where the MI_LOAD_REGISTER_MEMORY will read them.
    a. GPGPU_DISPATCHDIMX - the X dimension of the number of thread groups to dispatch in:

DWord	Project
7	CHV, BSW

    b. GPGPU_DISPATCHDIMY - the Y dimension of the number of thread groups to dispatch in:

DWord	Project
10	CHV, BSW

    c. GPGPU_DISPATCHDIMZ - the Z dimension of the number of thread groups to dispatch in:

DWord	Project
12	CHV, BSW

See vol1c Memory Interface and Command Stream for the MMIO register addresses and formats.

GPGPU Context Switch

The GPGPU pipeline supports interruption of GPGPU workloads on thread group boundaries. This is needed for general purpose GPGPUs that are so large that there is a risk of the display becoming non-responsive if the work cannot be interrupted for other jobs.

A workload is interrupted with the MI_ARB_CHECK command with the UHPTR register. The MI_ARB_CHECK command is placed throughout the command buffer. The driver updates the UHPTR register when a new context is needed; MI_ARB_CHECK checks for this and reprograms the head and tail pointers to the new batch of commands. The driver waits for the preemption to occur without going into RS2.

The GPGPU needs to modify this to allow a GPGPU_WALKER command to be interrupted. This is done by following each GPGPU_WALKER command with a MEDIA_STATE_FLUSH. This causes the CS to stop fetching commands until either the command completes or until the UHPTR valid bit is set.

GPGPU workloads can be dispatched with either GPGPU_OBJECT commands or GPGPU_WALKER commands. In the case of GPGPU_OJBECT, the MEDIA_STATE_FLUSH/ MI_ARB_CHECK pair must be placed in the batch buffer at thread group boundaries, since preemption cannot occur with a thread group partially dispatched. GPGPU_WALKER commands can dispatch multiple thread groups, in this case the MEDIA_STATE_FLUSH/ MI_ARB_CHECK follows each GPGPU_WALKER and the hardware takes care of noticing the UHPTR update and stopping at the next thread group boundary.

The commands in the batch buffer will look something like this:

Command Ring	Notes
MI_SET_CONTEXT	Go to GPGPU context
MI_BATCH_BUFFER_START	If new context, set address to top of batch. Otherwise, address needs to be set to the command preempted (given in the HWSP). The GP GPGPU bit must be set.

Command Batch	Notes
GPGPU_OBJECT	
GPGPU_OBJECT	
...	(more threads forming a complete thread group)
MEDIA_STATE_FLUSH	Check for preemption at thread group boundary. “Preemption” defined by the UHPTR valid bit set.
MI_ARB_CHECK	Move the head only if UHPTR valid bit is set.
...	
GPGPU_WALKER	
MEDIA_STATE_FLUSH	Check for preemption at thread group boundary internal to GPGPU_WALKER command. “Preemption” defined by the UHPTR valid bit set.
MI_ARB_CHECK	Move the head only if UHPTR valid bit is set.
...	
MI_BATCH_BUFFER_END	GPCS batch workload bit is cleared.

The context saved will consist of the state commands for VFE and a modified GPGPU_WALKER command with a new starting thread group id. On context restore, the commands are executed to start the GPGPU_WALKER where it left off before continuing with the rest of the command buffer.

An example software model for starting a preemption goes like this:

1. The UHPTR is reprogrammed to point to the current tail of the ring buffer.
2. Insert new commands:
    a. LRI to UHPTR to clear valid.
    b. Store Register to mem the preempted batch offset.
    c. Store Register to mem the preempted ring offset.
    d. Pipe_control notification.
    e. An MI_SET_CONTEXT to the new context is put into the ring.
3. Insert commands for new context. i.e. batch buffers.
4. Update Tail Pointer.

Programming Note	
Context:	GPGPU Context Switch
• 2-3 items above could happen during execution of a thread group so the HW may see the tail pointer updated before preemption starts. • The driver needs to turn off RC6 during items 1 and 4.	

GPGPU_CSR_BASE_ADDRESS

GPGPU Context Switch

Context switch allows the switch to take place in the middle of a thread group to provide better response time.

The command sequence has been simplified – MEDIA_STATE_FLUSH and MI_ARB_CHECK are now optional between commands for preemption to occur. MEDIA_STATE_FLUSH is now only needed before MEDIA_LOAD_CURBE commands to ensure CURBE is done being read before reloading it. The watermark bit in MEDIA_STATE_FLUSH is not needed, since the check is done automatically before a thread group is started.

Preemption can occur on commands listed here:

- MI_ARB_CHECK
- MEDIA_STATE_FLUSH
- PIPE_CONTROL
- MI_WAIT_FOR_EVENT
- MI_SEMAPHORE_WAIT
- GPGPU_WALKER – Preemption can occur at any time, with thread groups partially complete; the system state is saved/restored for context save and restore
- MI_WAIT_FOR_EVENT
- MI_SEAMPHORE_WAIT

Media commands like MEDIA_OBJECT and MEDIA_OBJECT_WALKER are not preemptable.

Messages to the Sampler must use headers (controlled by bit 19 of the Message Descriptor) when pre-emption is enabled.

See GPGPU Context Switch Workarounds for information about programming restrictions and workarounds.

Note that command preemption is not supported for MEDIA_OBJECT_* commands – MI_ARB_ON/OFF should be used to prevent preemption except at frame boundaries, where an MI_ARB_CHECK should be inserted.

The memory map of the context image that is saved from GPGPU pipeline includes SLM and EU State. A contiguous space is allocated to save the SLM and EU State starting from the address provided in GPGPU_CSR_BASE_ADDRESS.

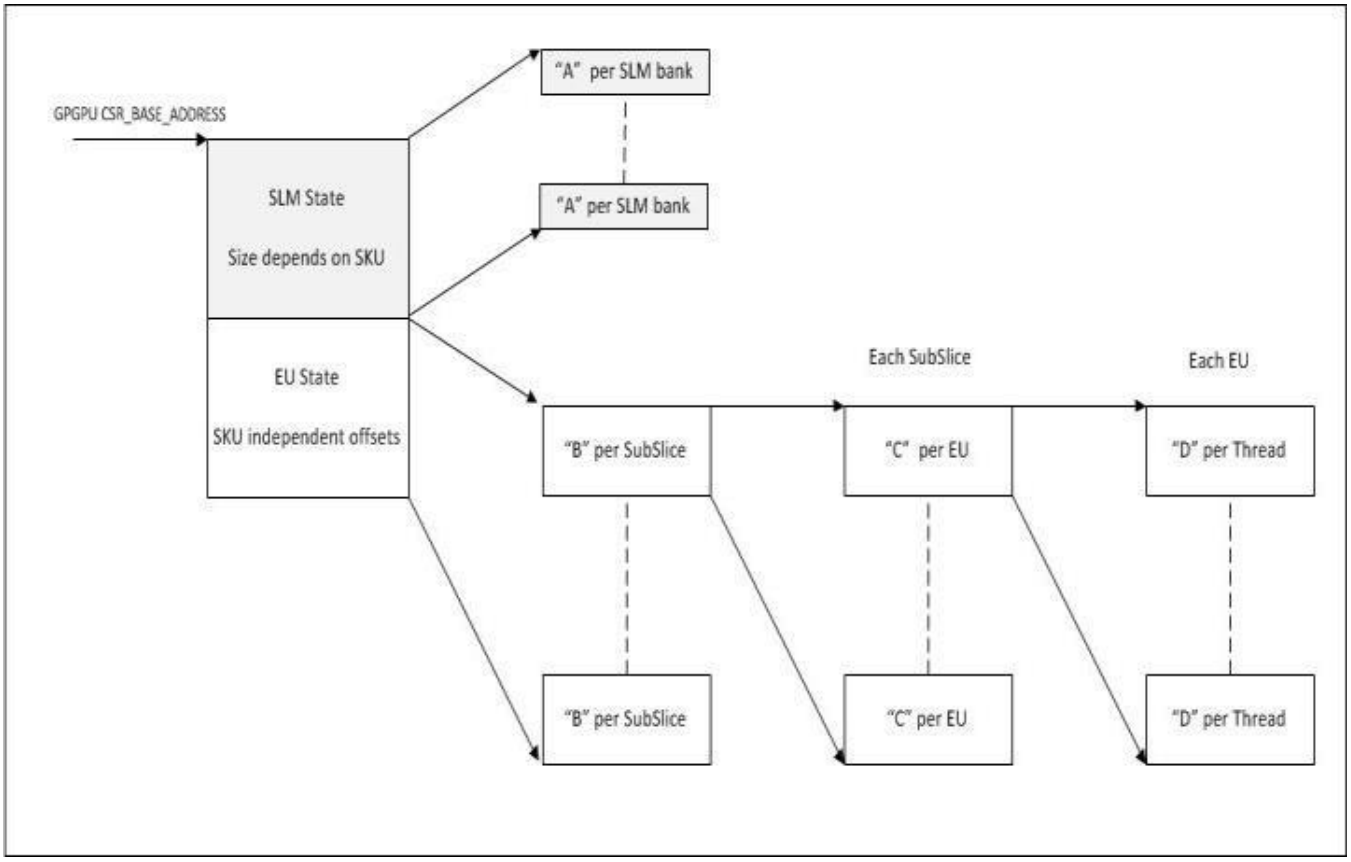
Maximum Upper Bound

Project	Description
CHV, BSW	The maximum upper bound is set to 4MB for GT2 and GT3.

In GPGPU context save/restore mode, hardware writes to this location and does NOT use the surface state or scratch space.

Memory Map of the GPGPU Context Image

The address offsets computed by hardware depend on the number of slices, subslices, EUs, and threads.



Base Address Calculation

Base Address = CSR_Base_Address + A* [Config.NumSlmBanks] + B*SubSliceId + C*EuId + D*ThreadId + Message_Offset

A = 0x10000 // 64K SLM per SubSlice

B = C * [Config.NumEusPerSubSlice]

C = D * [Config.NumThreadsPerEu]

D = 0x2000 // 8KB

Note: The slicenumber and EUIDs may require re-mapping such that a contiguous space is used with no gaps inbetween.

GPGPU Context Switch Workarounds

The GPGPU Context Switch support has some subtle programming restrictions and workarounds, as described below.

In products with slices that have an unequal number of EUs (for example, 1 slice with 23 EUs and 2 slices with 24) there will sometimes be a delay in pre-emption while the hardware waits for threads to exit in the smaller slice.

Media GPGPU Payload Limitations

There are 3 types of payload that the media/GPGPU instructions can have, but not all of them are allowed. The following table lists the legal combinations:

Workload	Commands	Data Stored
GPGPU	GPGPU_WALKER	CURBE
	GPGPU_WALKER	INDIRECT
Media(Legacy)	Media_Object	CURBE
	Media_Object	INDIRECT
	Media_Object	INLINE
	Media_Object	CURBE+INLINE
	Media_Object	CURBE+INDIRECT
	Media_Object	INLINE+INDIRECT
	Media_Object	CURBE+ INLINE+INDIRECT
	Media_Object_Walker	CURBE
	Media_Object_Walker	INLINE
	Media_Object_Walker	CURBE+INLINE
Media using Barrier/SLM	Media_Object_GRPID	CURBE
	Media_Object_GRPID	INDIRECT
	Media_Object_GRPID	INLINE
	Media_Object_Walker (with group id)	CURBE
	Media_Object_Walker (with group id)	INLINE

Indirect and CURBE payloads are fetched during thread dispatch from memory using the Dynamic State MOCS (specified in **[Instruction] STATE_BASE_ADDRESS**).

Synchronization of the Media/GPGPU Pipeline

The Media/GPGPU Pipeline is synchronized in the same way as the 3D pipeline using the PIPE_CONTROL command.

See the Bspec section on 3D pipe synchronization: vol2a 3D Pipeline - Overview [CHV, BSW] > 3D Pipeline > Synchronization of the 3D Pipeline.

Mode of Operations

This section contains registers for GPGPU Object and GPGPU Command. It also covers GPGPU Mode.

GPGPU Thread R0 Header

The RO header of the Thread Dispatch Payload for the GPGPU thread:

DWord	Bits	Description
R0.7	31:0	**Thread Group ID Z:** This field identifies the Z component of the thread group that this thread belongs to.
R0.6	31:0	**Thread Group ID Y:** This field identifies the Y component of the thread group that this thread belongs to.
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1K-byte aligned pointer to the scratch space (used for the GPGPU local memory space). Format = GeneralStateOffset[31:10]
	9	**GPGPU Dispatch.** Indicates that this dispatch is from the GPGPU pipe (see PIPELINE_SELECT command).
	8:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent threads (of any thread group). It is used to free up resources used by the thread upon thread completion. Format = U9.
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4	Reserved: MBZ
	3:0	Indicates the stack memory size. Range = [0,11] indicating [1K bytes, 2M bytes] **Programming Note**: Exception handling on stack overflow is not supported when GPGPU mid-thread pre-emption is desired.
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 16-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space.

DWord	Bits	Description
		Format = U4 Range = [0,10] indicating [2K bytes, 2M bytes] in powers of two.
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 16-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two.
	31	Reserved: MBZ
R0.2	30	Reserved: MBZ
	29	**Barrier Enable:** This field indicates that a barrier has been allocated for this kernel.
	28	**SLM Enable:** This field indicates that Shared Local Memory has been allocated for this kernel.
	27:24	**BarrierID:** This field indicates the barrier that this kernel is associated with. Format: U4
	23:15	Reserved
	14:10	Reserved: MBZ
	3:0	Reserved: MBZ
R0.1	31:0	**Thread Group ID X:** This field identifies the X component of the thread group that this thread belongs to.
R0.0	31:30	**Reserved:** MBZ
	29:28	Reserved: MBZ
	27:24	**Shared Local Memory Index:** Indicates the starting index for the shared local memory for the thread group. Each index points to the start of a 4K memory block, 16 possibilities cover the entire 64K shared memory per half-slice. Format = U4
	23:16	Reserved: MBZ
	15:0	**URB Handle:** This is the URB handle indicating the URB space for use by the thread.

Cross-thread CURBE if present is in R1 and above, followed by the X/Y/Z thread id values for each channel in the thread.

Project	Description
CHV, BSW	The GPGPU_WALKER command is supported.
CHV, BSW	At least one form of payload (either indirect or CURBE) must be sent with the GPGPU_WALKER.

GPGPU_WALKER

GPGPU Mode

The general purpose (GPGPU) mode allows the Gen7 architecture to be used by general purpose parallel APIs:

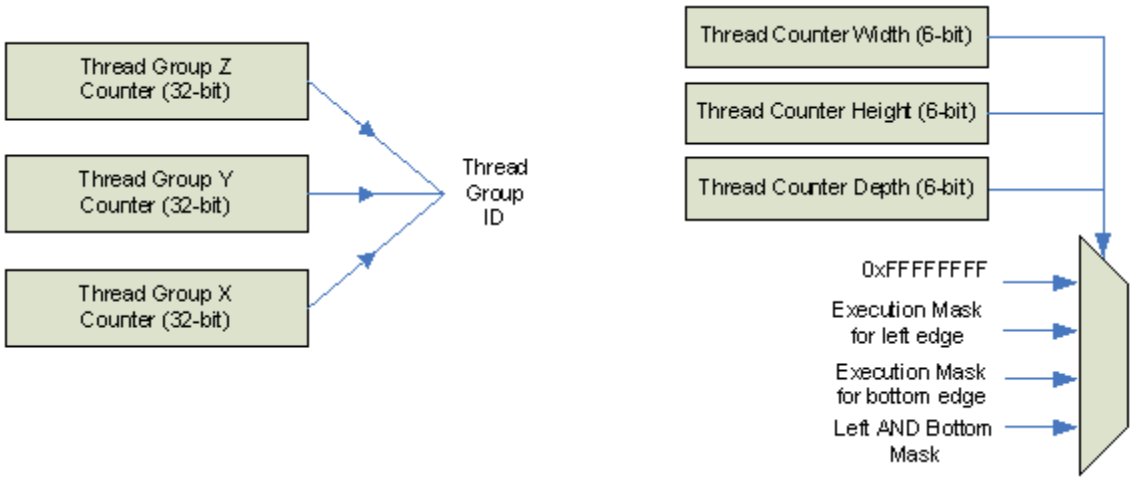
- GPGPU
- DX11 GPGPU

This is similar to the Generic mode with additional support for automatic generation of threads, Shared Local Memory, and Barriers.

Automatic Thread Generation

A single GPGPU job may require thousands or even millions of GPU_OBJECT commands. Rather than create them separately, it would be better to generate them algorithmically. To do this a GPGPU_WALKER command is created.

Rather than modifying the Media Walker, a simple Thread Group Walker is created instead:



The X/Y/Z counters for the thread group will have an initial and maximum value. The thread group ID sent with each dispatch consists of these 3 numbers. These counters are 32 bits since the spec does not limit the size of the thread ID.

The 3 thread counters count the number of dispatches in a single thread group – up to 32 dispatches for SIMD32 or 64 dispatches for SIMD16/8. There are 3 thread counters in order to select the execution masks correctly – see the section *Execution Masks*. Each one is 6 bits to allow full flexibility of any dimension going to 64 while the rest do not increment.

A thread is generated each time one of the thread counters increment. When all the counters reach their maximum values, the thread group is done and the thread group counter can increment and start a new thread group. When the thread group X counter reaches its maximum it is reset to 0, and the Y counter is incremented.

The compiler determines how many SIMD channels are needed per thread group, and then decides how these are split among EU threads. The number of threads is programmed in the thread counter, and the SIMD mode (SIMD8/SIMD16/SIMD32) is specified in the GPGPU_WALKER command.

Project	Security	Description
CHV, BSW	WA.Restriction	The maximum thread group size must fit into a single subslice and run in parallel, so the number of EU threads must be less than the number specified in Configurations for threads per subslice.
CHV, BSW	WA.ProgNote	Some tables do not have this broken out separately. The threads per subslice = (Thread count) / (Slice Count * Samplers).

Thread Payload

The payload to each thread dispatched is:

1. A thread group id which identifies the group the set of threads belong to. This is in the form of a set of 3, 32-bit X/Y/Z values.
2. The set of X/Y/Z that form the thread ID for each channel. If Z is not used then only X/Y are needed.
3. The execution mask which indicates which channels are active.

Thread IDs form a 2D or 3D surface which has to be mapped into SIMD32, SIMD16 or SIMD8 dispatches. Rather than have the hardware force a particular mapping of thread IDs to channels, the mapping will be supplied by the compiler. The VFE will receive a simple count of the number of threads per thread group which will be used to count the number of dispatches. The thread IDs for all threads in a thread group are put in a constant buffer with the MEDIA_CURBE_LOAD command. A single set of thread IDs can be used repeatedly for all thread groups, since the thread IDs are the same for each thread group ID output by the GPGPU_WALKER.

The data required is up to the compiler, but here is an example set of payloads for a 2 Z x 2Y x 12 X and a SIMD16 dispatch. This thread group requires 3 dispatches:

3 2 1 0 11 10 9 8 7 6 5 4 3 2 1 0	Thead id X for dispatch 0
1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0	Thead id Y for dispatch 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	Thead id Z for dispatch 0
7 6 5 4 3 2 1 0 11 10 9 8 7 6 5 4	Thead id X for dispatch 1
0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1	Thead id Y for dispatch 1
1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0	Thead id Z for dispatch 1
11 10 9 8 7 6 5 4 3 2 1 0 11 10 9 8	Thead id X for dispatch 2
1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0	Thead id Y for dispatch 2
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Thead id Z for dispatch 2

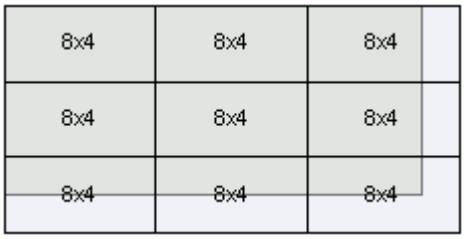
In this case the thread counter width would be programmed with a maximum value of 3 (since all the execution masks are all F, it doesn't matter how the thread counters are programmed as long as they count to 3 before finishing the thread group).

The first dispatch would tell the TS (who would tell the TD) that the payload starts at the beginning of the constant buffer and has a length of 3. The next dispatch would have a payload starting at constant_buffer_start + 3. The final dispatch payload starts at constant_buffer_start + 6. If there are more thread groups in the command they would get exactly the same payload – the only difference is the thread group ID (as well as a different barrier and shared local memory space).

Execution Masks

The number of channels required by the GPGPU job may not evenly fit into the number of SIMD channels. That can leave some channels idle. The execution mask is used to tell the hardware which channels are to be used.

A thread group is modeled as a 3D solid with each channel acting as one X/Y/Z point in the solid. This can take the form of a line with 1024 channels with X from 0 to 1023 and constant Y/Z, a square with X=0 to 32 and Y=0 to 32, or a cube with X=0 to 9, Y=0 to 9, Z=0 to 9. Software needs to determine how these shapes are mapped onto the 32 SIMD32 channels per dispatch (or 16 SIM16, etc). The mapping per thread is assumed to be a 2D square of channels such as 8x4, 16x2, 32x1. Below is a diagram of a 22x6 thread group that is mapped onto a set of 8x4 SIMD32 channels:

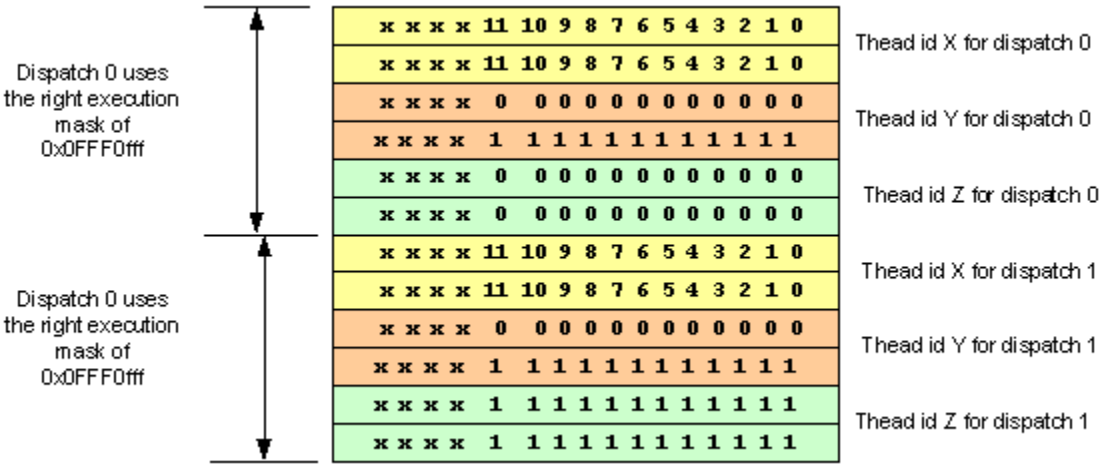


Note that the dispatches to the top and left have execution masks of all-F, while all the right edge dispatches have the same execution mask; likewise all the bottom edge dispatches have the same execution mask. The bottom right is the logical-AND of the right and bottom edge dispatches.

A 32-bit right and bottom mask is sent with the GPGPU_WALKER command, and the thread width, height and depth counters are used to determine when they are used (width, height and depth are used instead of X/Y/Z, since it is not required that width = X – width and height are the two variables that are changing in a single SIMD dispatch even if they are Y and Z).

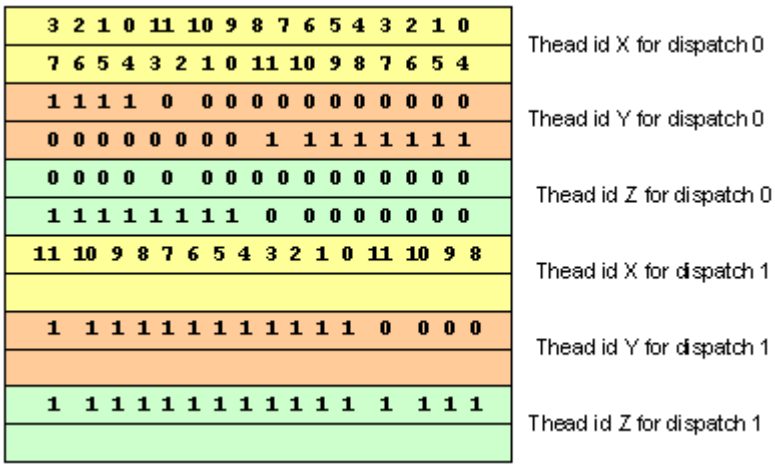
For each dispatch the width counter is incremented until it reaches the maximum – the dispatch with width=max will use the right execution mask. The height counter is then incremented and process repeated. If at any time the height counter = max then the execution mask is the bottom execution mask. When the height and width counters are both max then the dispatch will be the AND of the right and bottom and the depth counter will increment.

The same 2Z x 2Y x 12X thread group described above dispatched as SIMD32 with each dispatch delivering a 16X x 2Y shape would require 2 dispatches with empty bits in the right execution mask and all F in the bottom.



The width and height counter would have a maximum of 1, and the depth counter would have a maximum of 2. The two dispatches would use the AND of the two masks, but since the bottom mask is F it would be the same as just the right mask.

The execution masks can also be used when the software wants to pack the channels rather than lay them out in a regular pattern:



In this case the width counter can have a maximum of 2, and the height and depth counters with a maximum of 1. The first dispatch will use the bottom mask only (all-F) and the second will use the right AND bottom mask to remove the channels that are not used.

URB Management

The VFE manages the URB in GPGPU and generic/media modes.

Description
The first 64 URB entries are reserved for the interface description, and CURBE data is placed after the IDs. URB handles are needed for indirect data and parent/child communication; when the VFE starts up it creates up to 128 handles by partitioning the remaining URB space into evenly spaced addresses and saving the resulting handles in a FIFO. The handles can then be treated just like ones created by the URBM - send to TD on dispatch and

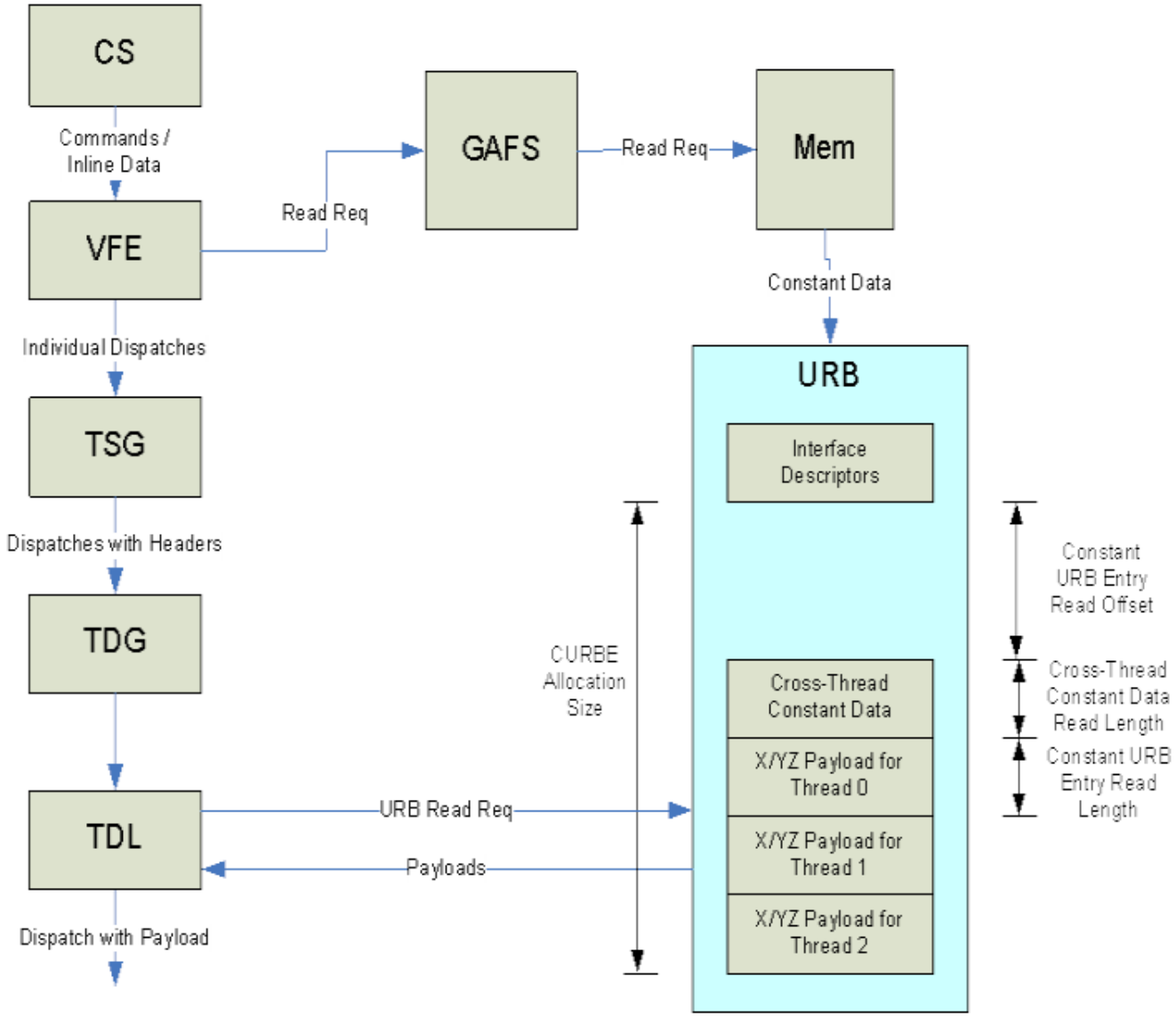
Description
recovered on the handle return bus. MEDIA_VFE_STATE specifies the amount of CURBE space, the URB handle size and the number of URB handles. The driver must ensure that: ((URB_handle_size * URB_num_handle) – CURBE – 64) <= URB_allocation_in_L3.

Indirect Payload Storage

The GPGPU commands are extended to allow indirect input as an alternative to CURBE. The mechanism used for CURBE will be used in exactly the same way for indirect: the same offset is used which specifies what data is delivered to all threads, and then a count which specifies how much data is delivered per thread. A single indirect pointer points to both the Cross-Thread and Per-Thread Constant Data, which is stored in the URB. The position of the Cross-Thread and Per-Thread constants are swapped in the EU GRF.

To use indirect payload storage, the (URB Entry Allocation Size * Number of URB Entries) product must be enough to cover the sum of the Cross-Thread and Per-Thread Indirect data to be loaded. These state variables are set in the MEDIA_VFE_STATE command. The URB Entry Allocation must be equal or greater than th GPGPU_WALKER's Indirect Data Length. In general the maximum number of URB Entries that fills the URB space should be used to ensure that a shortage of handles doesn't cause a performance problem.

CURBE Payload



Example of CURBE command sequence:

MEDIA_STATE_FLUSH // Make sure dispatch is not accessing CURBE

MEDIA_CURBE_LOAD

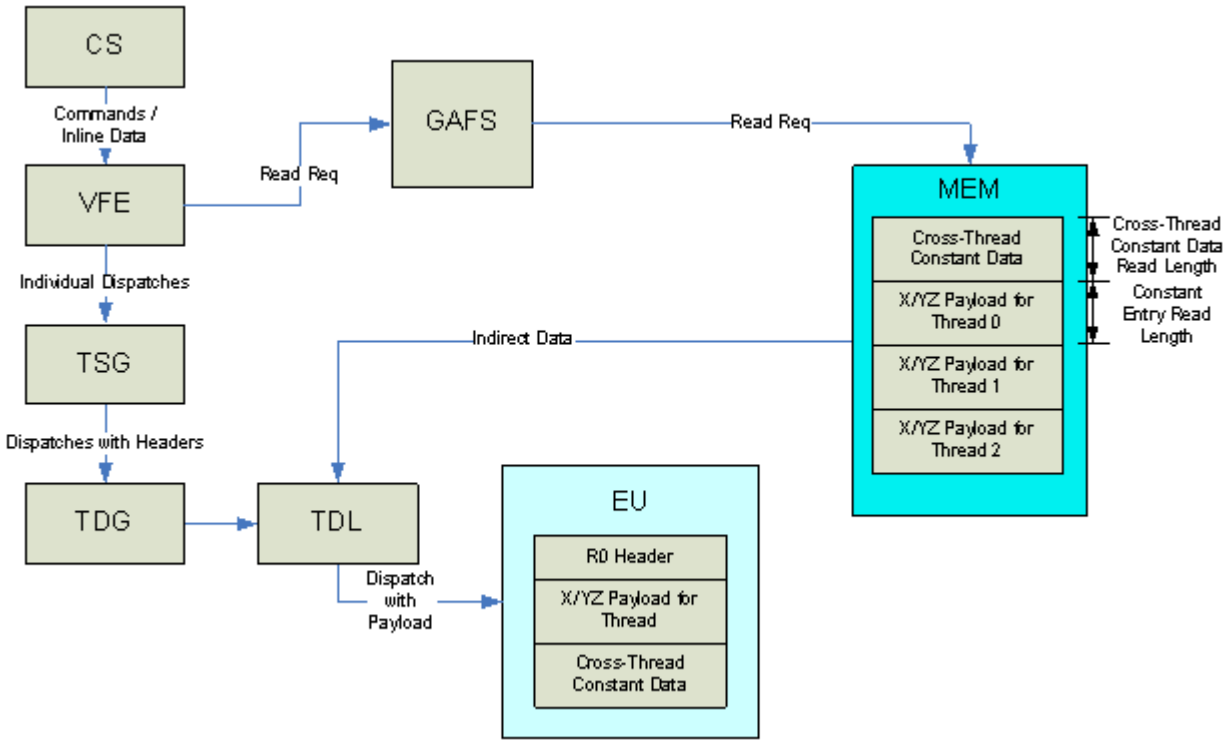
GPGPU_WALKER

MEDIA_STATE_FLUSH

MEDIA_CURBE_LOAD

GPGPU_WALKER

Figure for Indirect Payload



Example of Indirect Command Sequence:

GPGPU_WALKER // Indirect pointer included in command

GPGPU_WALKER // No need to flush between commands for CURBE.

The differences between CURBE and Indirect are:

- The indirect uses a 32-bit memory pointer for the start address.
- The Constant URB Entry Read Offset is not used.
- The Cross-Thread Constant Data Read Length and the Constant Entry Read Length is multiplied by 32 to convert it into bytes.
- The X/Y/Z payload in the EU GRF comes before the Cross-Thread Constant Data.

MEDIA_OBJECT_GRPID

GPGPU_OBJECT already allows barriers and SLM to be allocated, but does not allow the scoreboard to be used. We need a single command which allows both.

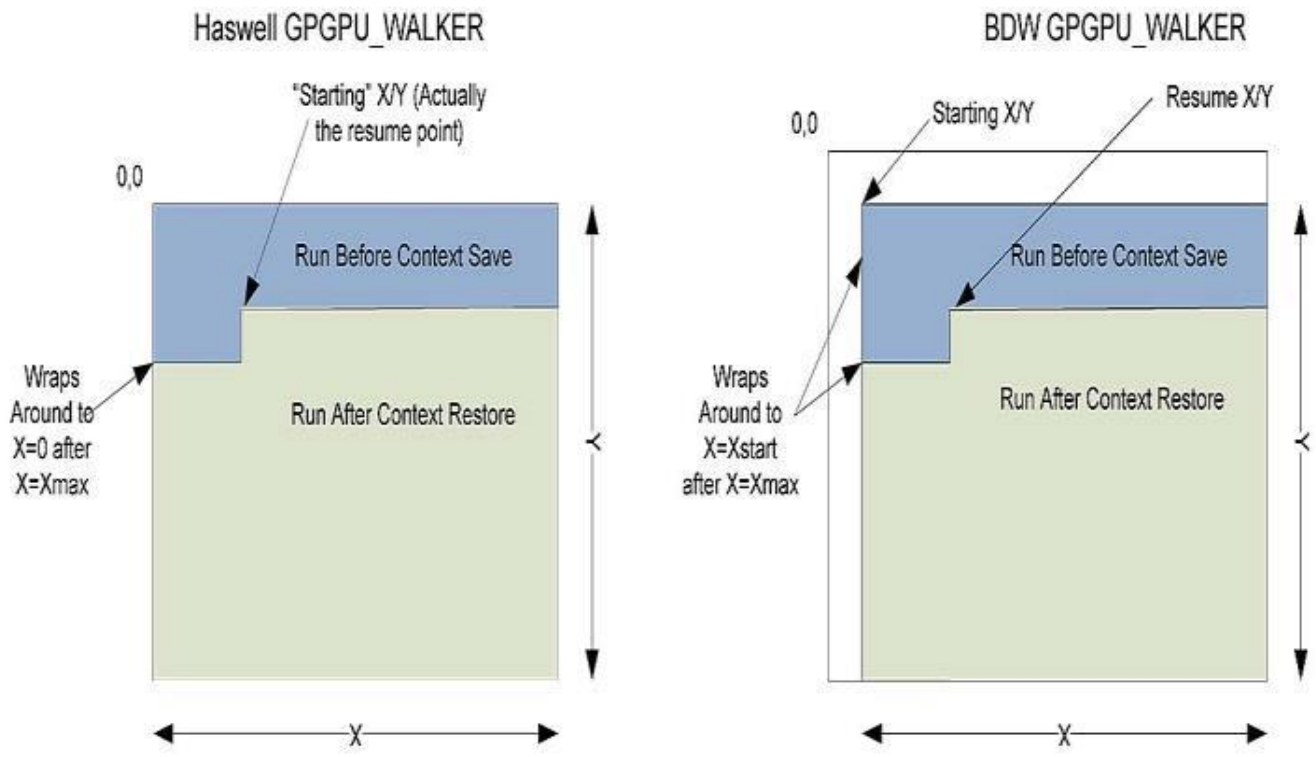
A new command called MEDIA_OBJECT_GRPID is created which is the same as the basic MEDIA_OBJECT command with the addition of a 32-bit group id and an end-of-group bit. The group id will be used to allocate barriers and SLM based on the Interface Descriptor.

Restrictions that are currently in place for GPGPU would carry over to Media use of these features:

- All threads that use them must be on the same Tslice unless cross-slice barriers are enabled.
- We only dispatch a single group at a time – this is naturally supported by using only the most outer loop numbers for the in MEDIA_OBJECT_WALKER, but must be enforced by the programmer for MEDIA_OBJECT_GRPID.

Starting Offset for a Thread Group ID

The GPGPU_WALKER command has starting values for the Thread Group IDs that the walker generates per thread group, but they are designed to allow context save and restore rather than provide a true initial value. When walking a 2D set of groups in X/Y, when the X value reaches the maximum, the Y increments and the next X is 0, rather than going back to Xstart. This behavior is expected on context restore where the true initial values are 0,0 and the starting values are actually the resume values of the restore.



Thread Group Tracking

The TSG needs to keep track of the threads outstanding in a group to know when the thread group barrier and Shared Local Memory can be reclaimed. This can be done by keeping a counter per active thread group (up to 16 per half-slice) which increments when a new thread is sent out and decremented when the thread retires. The assigned barrier ID (with half-slice bit) is unique per thread group and much smaller than the thread group ID and so will be used to keep track of the thread group instead.

Since TSL sends the thread retirement via the Message Channel rather than the thread retirement bus, the barrier ID used to identify the thread group can be sent at the same time. A CAM will then match the ID with the counter to decrement.

There is a potential corner case of a thread group without barriers being partly dispatched, then retiring before the rest of the thread group is sent. This should be OK, since the lack of barriers means that there are no dependencies between threads.

Shared Local Memory Allocation

The Shared Local Memory is a 64k block per half-slice in the L3 that must be shared between all thread groups on that half-slice. A new memory manager simular to the Scratch Space memory manager is used to allocate this space.

We are only dispatching threads from a single Interface Descriptor at a time. If a new Interface Descriptor is requested the pipe is drained and all shared memory recovered before starting to allocate new shared memory. This means that only a single size of shared memory needs to be supported at once.

For simplicity, only power-of-2 sizes from 4k to 64k are allowed. The thread request will specify how much is needed. The first thread of a Thread Group is marked as requiring a new shared local memory – if not the old Shared Local Memory offset is sent with the dispatch.

A simple set of 16-bits is used to allocate 4k shared memory, with fewer bits used for larger sizes. A priority encoder finds the first unused bit and the offset remembered as being associated with a particular barrier id. The barrier id is then used to track the thread group.

When the Thread Group Tracking indicates that a thread group is completely retired, that section of shared local memory can be reclaimed.

Software Managed Shared Local Memory

Software can optionally manage shared local memory. In this case, each thread command or thread group command will have the shared memory offset included – each command in a thread group must have the same offset, of couse. If the offset requested is still being used then the command is stalled until the thread group using that offset is done.

Hardware will track the usage of this section of shared memory as before, recording the offset as being used and recording it as being available after the thread group is done.

Automatic Barrier Management

Description
Since we have an automatic shared memory allocation it makes sense to make barrier management automatic too. Instead of the barrier id in the Interface Descriptor, there is now a thread count per thread group.
If a new thread group id comes in without a barrier allocated (checked with a CAM match across 16 barriers), the TSG picks a unused barrier and sends this count in a message to GWunit. It then needs to wait for an accept message back from GW before sending the dispatch to ensure that a barrier message doesn't arrive at the GW before the barrier is programmed. The barrier ID picked is sent with every dispatch from this thread group.
When the thread group tracker determines that a thread group has finished, the barrier becomes available to new thread groups.

Dispatch Payload

The payload for a general purpose thread will have to include the execution mask with a bit per 32-channel. SIMD16 and SIMD8 use the LSB bits of the execution mask. The 5-bit number transferred from VFE will be expanded to produce the 32-bit mask. This will use the Dmask currently used by the pixel shader dispatch in the transparent header.

Generic Media

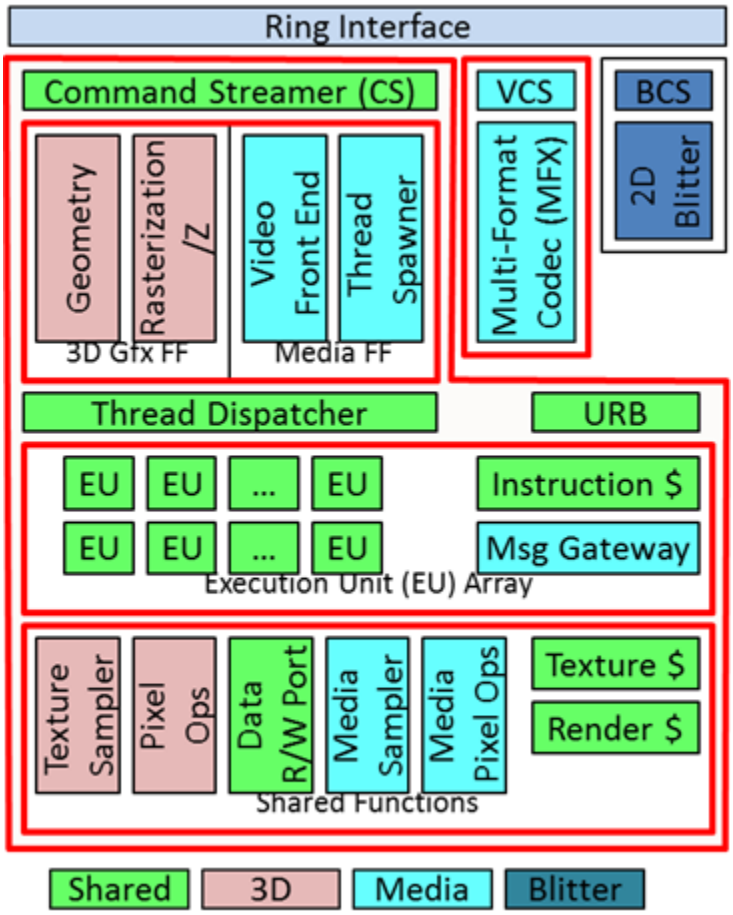
This introduction provides a brief overview of the Media product features. It includes Media functions, feature benefits, and how the features fit into graphics products as part of a whole solution.

Media product features include:

- Multi-format codec engine
- Video front end
- Media fixed functions
- Video encoding
- Video decoding
- Sampling

These product features support specific applications, such as interactive gaming, videogames, social media, virtual reality, and augmented reality.

The following block diagram shows the Main Render Engine, unified for 3D graphics and Media.

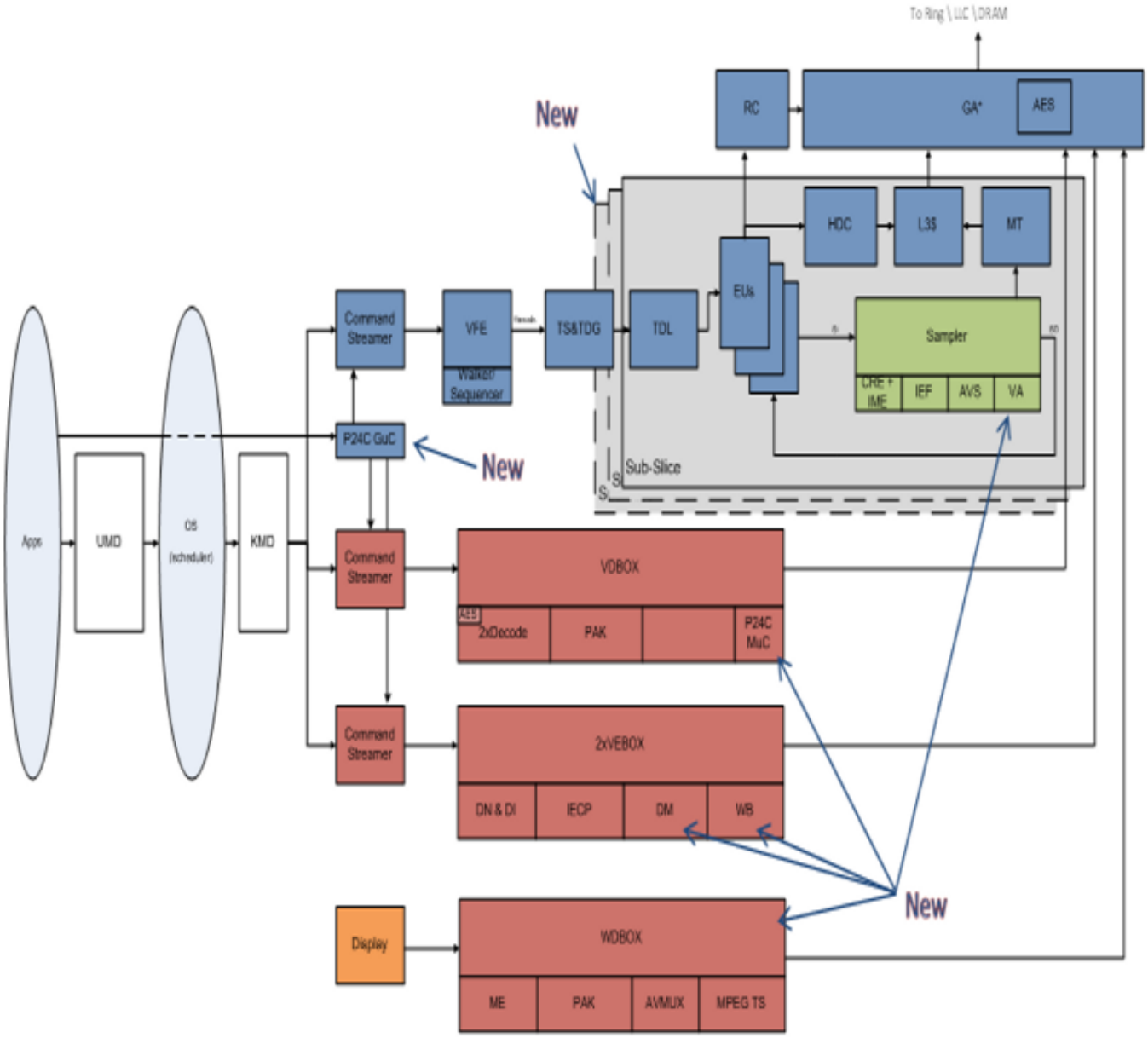


- **Fixed Function (FF) pipelines:** Provide thread generation and control.
- **3D graphics or Media FF:** Controls EU array at a given time. The EU (Execution Unit) array is shared between 3D and Media; ISA is optimized for both.
- **Shared functions:** Are accelerators to run filtered load, scatter, gather, and filter/blended store operations.
- **MFX:** Is a parallel codec engine that runs in a separate context.

Product Evolution

Block diagrams in this section describe the evolution of Media products, by project. They include definitions of the main components and how they integrate with each other.

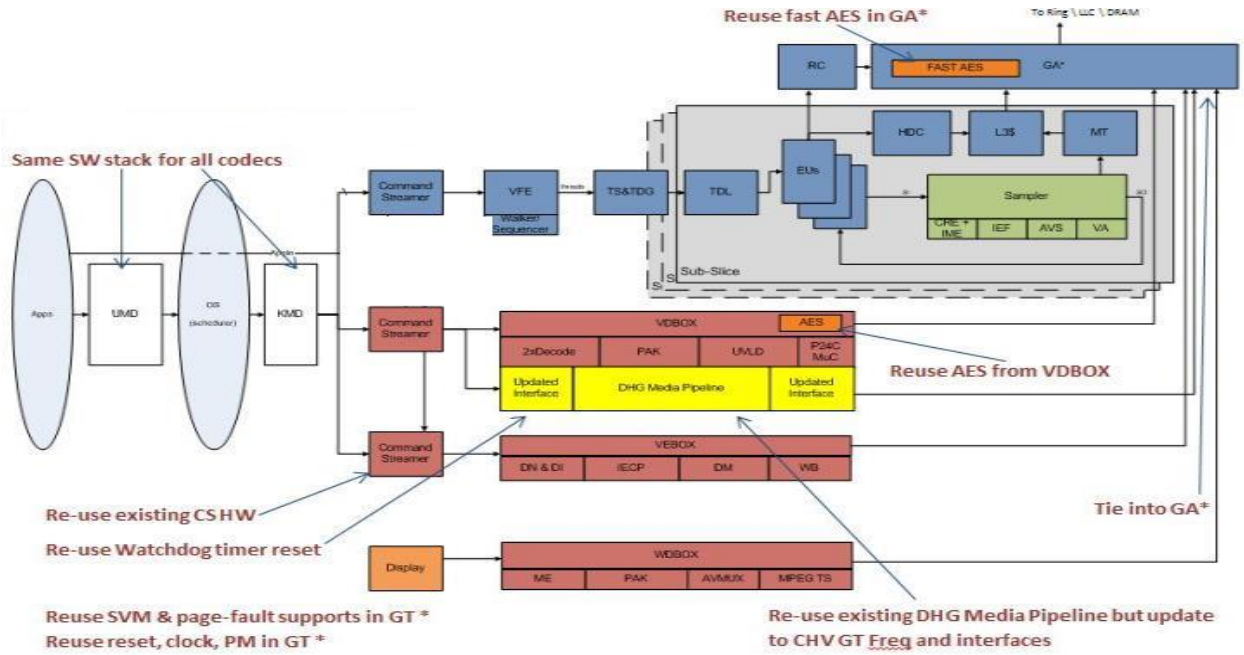
CHV, BSW GT2 Media Pipelines



Additions/Changes:

- **2xVEBOX:** Two instances of VEBOX on the same chip.
  - **DM:** De-Mosaic (image processing used to reconstruct a full-color image from sensor output, which is usually in Bayer format).
  - **WB:** White Balance is the process of removing unrealistic color casts.
- **VA:** Computer Vision/Analytics - Accelerators for commonly used mathematical operations for computer vision functions.
- **Decode HW:** AVC, MPEG-2, MVC, VC-1, VP8, JPEG and legacy formats (AVS, MPEG-4p2).
- **Encode HW:** AVC, MPEG-2, MVC, JPEG, and VP8.

Converged CHV, BSW (14nm) Solution



CHV, BSW GT Media Pipelines: This solution maximizes re-use of existing assets while providing legacy codec coverage.

- Re-use **HW Scheduler and Interrupt Handler**.
- Same stack for all **Codecs**.
- Re-use existing **CS HW**.
- Re-use **Watchdog Timer Reset**:
  - **Updated Interface**.
  - **DHG Media Pipeline:** IP from the Digital Home Group.
- Tie into **GA***.
- Re-use SVN and Page Fault supports in **GT***.
- Re-use Reset, Clock, PM in GT*.
- Re-use existing **DHG Media Pipeline,** but update to Frequency and Interfaces.

Media and General Purpose Pipeline

Introduction

This section covers the programming details for the media (general purpose) fixed function pipeline. The *media pipeline* is positioned in parallel with the 3D fixed function pipeline. It provides media functions and has media specific fixed function capability. However, the fixed functions are designed to have the general capability of controlling the shared functions and resources, feeding generic threads to the Execution Units to be executed, and interacting with these generic threads during run time. The media pipeline can be used for non-media applications, and therefore, can also be referred to as the *general purpose pipeline*. For the rest of this chapter, we refer to this fixed function pipeline as the media pipeline, keeping in mind its general purpose capability.

Concurrency of the media pipeline and the 3D pipeline is not supported. In other words, only one pipeline can be activated at a given time. Switching between the two pipelines within a single context is supported using the MI_PIPELINE_SELECT command.

Following are some media application examples that can be mapped onto the media pipeline. All these applications are functional; however, the level of performance that can be achieved depends on the hardware configuration and is beyond the scope of this document.

Application
MPEG-2 decode acceleration with HWMC (e.g. DXVA HWMC interface)
MPEG-2 decode acceleration with IS/IDCT and forward (e.g. DXVA IDCT interface)
MPEG-2 decode acceleration with VLD and forward (e.g. DXVA VLD interface)
AVC decode acceleration with HWMC and forward including Loop Filter
VC1 decode acceleration with HWMC and forward including Loop Filter
Advanced deinterlace filter (motion detected or motion compensated deinterlace filter)
Video encode acceleration (with various level of hardware assistance)

Terminologies

Term	Description
AVC	Advanced Video Coding. An international video coding standard jointly developed by MPEG and ITU. It is also known as H.264 (ITU), or MPEG-4 Part 10 (MPEG).
Child Thread	A thread corresponding to a leaf-node or a branch-node in a thread generation hierarchy. All thread originated from kernels running on the execution units are child threads.
EOB	End of Block. It is a 1-bit flag in the non-zero DCT coefficient data structure indicating the end of an 8x8 block in a DCT coefficient data buffer.
IDCT	Inverse Discrete Cosine Transform. It is the stage in the video decoding pipe between IQ and MC.
IQ	Inverse Quantization. It is a stage in the video decoding pipe between IS and IDCT.
IT	Inverse Integer Transform. It is the stage in AVC or VC1 video decoding pipe between IQ and MC.

Term	Description
MPEG	Motion Picture Expert Group. MPEG is the international standard body JTC1/SC29/WG11 under ISO/IEC that has defined audio and video compression standards such as MPEG-1, MPEG-2, and MPEG-4, etc.
MC	Motion Compensation. It is part of the video decoding pipe.
MVFS	Motion Vector Field Selection – a four-bit field selecting reference fields for the motion vectors of the current macroblock.
PRT	A persistent root thread in general stays in the system for a long period of time. It is normally a parent thread. Only one PRT is allowed in the system. Hardware is responsible for re-dispatching the incomplete PRT at context restore, and a PRT can continue operations from that previously left-over state.
Parent Thread	A thread corresponding to a root-node or a branch-node in thread generation hierarchy. A parent thread may be a root thread or a child thread depending on its position in the thread generation hierarchy.
Root Thread	A thread corresponding to a root-node in a thread generation hierarchy. In the general-purpose pipeline, all threads originated from VFE unit are root threads.
Synchronized Root Thread	A root thread that is dispatched by TS upon a 'dispatch root thread' message.
TS	Thread Spawner. It is the second (and the last) fixed function in the general-purpose pipeline.
Unsynchronized Root Thread	A root thread that is automatically dispatched by TS.
VFE	Video Front End. It is the first fixed function in the general-purpose pipeline.
VLD	Variable Length Decode. It is the first stage of the video decoding pipe that consists mainly of bit-wide operations. Hardware MPEG-2 VLD acceleration is supported in the VFE fixed function stage.

Hardware Feature Map in Products

The following table lists the hardware features in the media pipe.

Video Front End Features in Device Hardware

Features/ Device	[CHV, BSW]
Generic Mode	**Y**
Root Threads	**Y**
Parent/Child Threads	**Y**

Features/ Device	[CHV, BSW]
SRT (Synchronized Root Threads)	**Y**
PRT (Persistent Root Thread)	**Y**
Interface Descriptor Remapping	N
IS Mode (HW Inverse Scan)	N
VLD Mode (HW MPEG2 VLD)	N
AVC MC Mode	N
AVC IT Mode (HW AVC IT)	N
AVC ILDB Filter (in Data Port)	N
VC1 MC Mode	N
VC1 IT Mode (HW VC1 IT)	N
Stalling HW Scoreboard	**Y**
Non-stalling HW Scoreboard	**Y**
HW Walker	**Y**
HW Timer	**Y**
Pipelined State Flush	**Y**
HW Barrier	**Y**

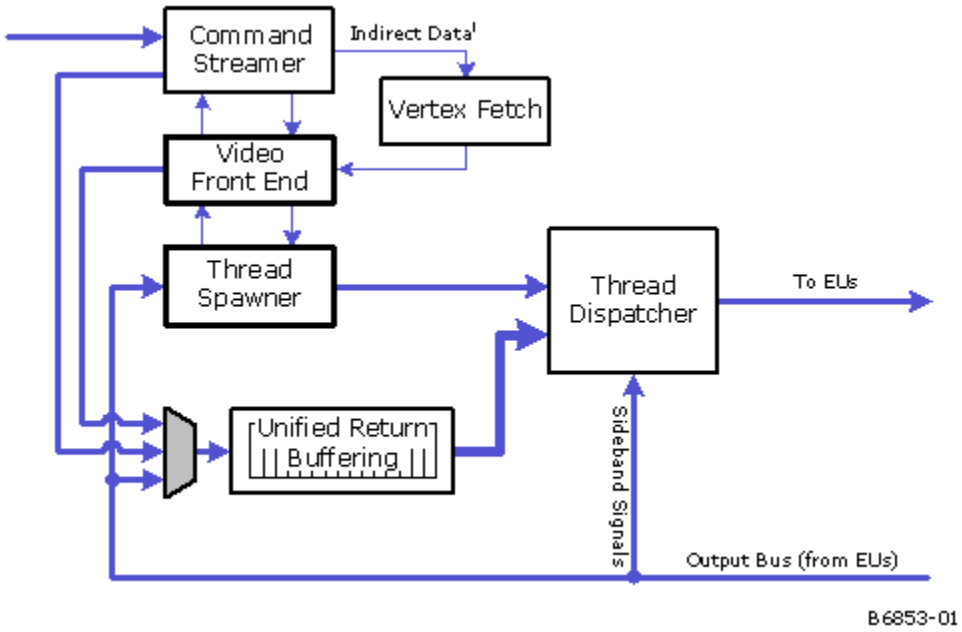
Media Pipeline Overview

The media (general purpose) pipeline consists of two fixed function units: Video Front End (VFE) unit and Thread Spawner (TS) unit. VFE unit interfaces with the Command Streamer (CS), writes thread payload data into the Unified Return Buffer (URB), and prepares threads to be dispatched through TS unit. VFE unit also contains a hardware Variable Length Decode (VLD) engine for MPEG-2 video decode. TS unit is the only unit of the media pipeline that interfaces to the Thread Dispatcher (TD) unit for new thread generation. It is responsible for spawning root threads (short for the root-node parent threads) originated from VFE unit and for spawning child threads (can be either a leaf-node child thread or a branch-node parent thread) originated from the Execution Units (EU) by a parent thread (can be a root-node or a branch-node parent thread).

The fixed functions, VFE and TS, in the media pipeline, in most cases, share the same basic building blocks as the fixed functions in the 3D pipeline. However, there are some unique features in media fixed functions as highlighted by the followings.

- VFE manages URB and only has write access to URB; TS does not interface to URB.
- When URB Constant Buffer is enabled, VFE forwards TS the URB Handler for the URB Constant Buffer received from CS.
- TS interfaces to TD; VFE does not.
- TS can have a message directed to it like other shared functions (and thus TS has a shared function ID), and it does not snoop the Output Bus as some other fixed functions in the 3D pipeline do.
- A root thread generated by the media pipeline can only have up to one URB return handle.
- If a root thread has a URB return handle, VFE creates the URB handle for the payload to initiating the root thread and also passes it alone to the root thread as the return handle. The root thread then uses the same URB handle for child thread generation.
- If URB Constant Buffer is enabled and an interface descriptor indicates that it is also used for the kernel, TS requests TD to load constant data directly to the thread's register space. For root thread, constant data are loaded after R0 and before the data from the other URB handle. For child thread, as the R0 header is provided by the parent thread, Thread Spawner splits the URB handles from the parent thread into two and inserts the constant data after the R0 header.
- A root thread must terminate with a message to TS. A child thread should also terminate with a message to TS.
- High streaming performance of indirect media object load is achieved by utilizing the large vertex cache available in the Vertex Fetch unit (of the 3D pipeline).

Top level block diagram of the Media Pipeline



Generic Mode

In the Generic mode, VFE serves as a conduit for general-purpose kernels fully configured by the host software. As there is no special fixed function logic used, the Generic mode can also be viewed as a 'pass-through' mode. In this mode, VFE generates a new thread for each MEDIA_OBJECT command. The payload contained in the MEDIA_OBJECT command (inline and/or indirect) is streamed into URB. The interface descriptor pointer is computed by VFE based on the interface descriptor offset value and the interface descriptor base pointer stored in the VFE state. VFE then forwards the interface descriptor pointer and the URB handle to TS to generate a new root thread. Many media processing applications can be supported using the Generic mode: MPEG-2 HWMC, frame rate conversion, advanced deinterface filter, to name a few.

Programming Media Pipeline

The Programming Media Pipeline is programmed with command sequences. The media hardware threads are created through the parameterized media walker. The dispatch of thread is controlled by a scoreboard mechanism.

Command Sequence

Media pipeline uses a simple programming model. Unlike the 3D pipeline, it does not support pipelined state changes. Any state change requires an MI_FLUSH or PIPE_CONTROL command. When programming the media pipeline, it should be cautious to not use the pipelining capability of the commands described in the Graphics Processing Engine chapter.

To emphasize the non-pipeline nature of the media pipeline programming model, the programmer should note that if any one command is issued in the "Primitive Command" step, none of the state commands described in the previous steps cannot be issued without preceding with a MI_FLUSH or PIPE_CONTROL command.

Note for [CHV, BSW]: With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. The MEDIA_STATE_FLUSH serves as a fence for state change by flushing the VFE/TS front ends but not waiting for threads to retire.

The basic steps in programming the media pipeline are listed below. Some of the steps are optional; however, the order must be followed strictly. Some usage restrictions are highlighted for illustration purpose. For details, refer to the respective chapters for these commands.

Command Sequence

For [CHV, BSW], the media pipeline is further simplified with fixed functions like MPEG2 VLD and AVC/VC1 IT removed. The addition includes:

1. The CURBE command is now unique to the media pipeline.
2. The interface descriptors are delivered directly as a media state command instead of being loaded through indirect state.

The programming model requires the following steps:

Step 1: MI_FLUSH/PIPE_CONTROL:

- This step is mandatory.
- Multiple such commands in step 1 are allowed, but not recommended for performance reasons.

Step 2: State command PIPELINE_ SELECT:

- This step is optional. This command can be omitted if it is known that within the same context the media pipeline was selected before Step 1.
- Multiple such commands in step 2 are allowed, but not recommended for performance reasons.

Step 3: State commands configuring pipeline states:

- STATE_BASE_ADDRESS:
  - This command is mandatory for this step (i.e. at least one).
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
  - This command must precede any other state commands below.
  - Particularly, the fields **Indirect Object Base Address** and **Indirect Object Access Upper Bound** are used to control indirect Media object load in VF.
  - The fields **Dynamics Base Address** and **Dynamics Base Access Upper Bound** are used to control indirect Curbe and Interface Descriptor object load in VF.
  - **Note:** This command may be inserted before (and after) any commands listed in the previous steps (Step 1 and 2). For example, this command may be placed in the ring buffer while the others are put in a batch buffer.
- STATE_SIP:
  - This command is optional for this step. It is only required when SIP is used by the kernels.
- MEDIA_VFE_STATE:
  - This command is mandatory for this step (i.e. at least one).
  - This command cause destruction of all outstanding URB handles in the system. A new set of URB handles will be generated based on state parameters, no. of URB and URB length, programmed in VFE FF state.
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_CURBE_LOAD:
  - This command is optional.
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.
- MEDIA_INTERFACE_DESCRIPTOR_LOAD:
  - This command is mandatory for this step (i.e. at least one).
  - Multiple such commands in this step are allowed. The last one overwrites previous ones.

Step 4: Primitive commands:

- MEDIA_OBJECT:
  - This step is optional, but it does not make practical sense to not issue media primitive commands after going through the previous steps to set up the media pipeline.
  - Multiple such commands in step 4 can be issued to continue processing media primitives.

With the addition of MEDIA_STATE_FLUSH command, pipelined state changes are allowed on the media pipeline. To support context switch for barrier groups, watermark and barrier dependencies are added to the MEDIA_STATE_FLUSH command. The usage of barrier group may have strict restriction that all threads belonging to a barrier group must all be present to avoid deadlock during context switch. Here are the example programming sequences to allow context switch.

Project	Notes
CHV, BSW	The use of MEDIA_OBJECT_PRT and MI_ARB_ON_OFF are optional.

- MEDIA_VFE_STATE
- MEDIA_INTERFACE_DESCRIPTOR_LOAD
- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 1)
- MEDIA_OBJECT_PRT (with VFE_STATE_FLUSH set and PRT NEEDED set)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 1
- Several MEDIA_OBJECT command (for barrier group 1)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- MEDIA_STATE_FLUSH (optional, only if barrier dependency is needed)
- MEDIA_INTERFACE_DESCRIPTOR_LOAD (optional)
- MEDIA_CURBE_LOAD (optional)
- MEDIA_GATEWAY_STATE (for example for barrier group 2)
- MEDIA_STATE_FLUSH (with watermark set for group 1)
- MI_ARB_ON_OFF (OFF) // Arbitration must be turned off while sending objects for group 2
- Several MEDIA_OBJECT command (for barrier group 2)
- MI_ARB_ON_OFF (ON) // Arbitration is allowed
- ...
- MI_FLUSH

Commands for the GPGPU pipe (GPGPU_OBJECT and GPGPU_WALKER) should be separated from commands for the Media pipe (MEDIA_OBJECT*) by an MI_FLUSH.

Parameterized Media Walker

The Parameterized Media Walker is a hardware thread generation mechanism that creates threads associated with units in a generalized 2-dimensional space, for example, blocks in a 2D image. With a small number of unit step vectors, the walker can implement a large number of walking patterns as described hereafter. This command may provide functions that are normally handled by the host software, thus, may be used to simplify the host software and GPU interface.

The walker described herein is doubly nested, where essentially a “local” walker can perform a variety of 2-dimensional walking patterns and a “global” walker can perform similar 2-dimensional walking patterns upon many local walkers. The local walker has 3 levels (outer, middle, and inner) while the global walker has 2 levels (outer and inner). Thus, the algorithm has 5-nested loops that modify local state based on user-defined unit step vectors.

The Walker’s programmability is derived from:

- The walker traverses a unit-normalized surface. Some example unit sizes:
    - 1x1: Walking pixels
    - 4x4: Walking sub-blocks

- 16x16: Walking macro-blocks
- 32x16: Walking macro-block-pairs

- The use of unit step vectors to describe the motion at each of level of nesting
- Starting locations for the local and global walkers
- Block sizes of the local and global walker
- And a small number of special mode controls for the inner-most loop which are aimed at efficiently dividing an image into two balanced workloads for dual-slice designs.

Walker Parameter Description

The global and local loops are both described by the same four parameters:

- Resolution
- Starting location
- Outer unit vector
- Inner unit vector

The local inner loop has some special modes that will be described later. A table of the user inputs and some example values are given below:

GLOBAL LOOP PARAMETERS							
Global Resolution		Global Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
120	68	0	0	32	0	0	32

LOCAL LOOP PARAMETERS							
Block Resolution		Local Start		Outer Loop Unit Vector		Inner Loop Unit Vector	
X	Y	X	Y	X	Y	X	Y
32	32	0	0	1	0	-2	2

LOCAL INNER LOOP SPECIAL MODE SELECTS							
Dual Mode	Repel	Attract			ExtraSteps	X	Y
TRUE	FALSE	FALSE			1	0	1

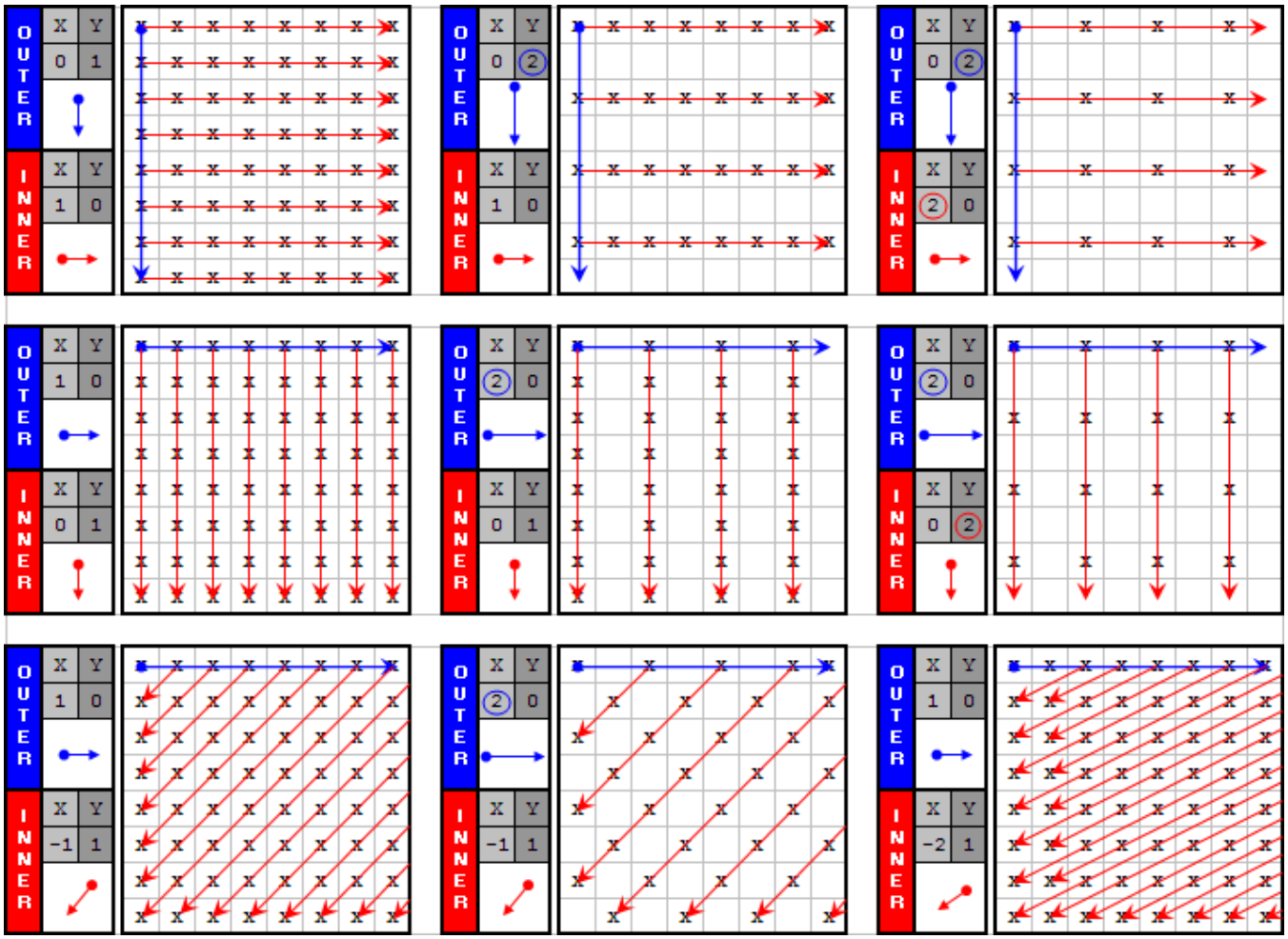
It should be emphasized that the value of what a "unit" represents is implicitly defined by the user. In other words, the walker traverses a "unit normalized space" that is not inherently bound to pixel walking. If the smallest unit of work the user wants to walk is a 4x3 block of pixels, you can program the inner loop to step (4,3) or (1,1):

- In the first case (4,3) the user is walking in units of pixels
- In the second case (1,1) the user is walking in units of 4x3 blocks of pixels.

It should be noted that hardware doesn't contain enough bits for pixel walking for pixel resolution like 1920x1088. The intended usage of the walker is for block walking whereas the block size is not relevant to the walker parameters.

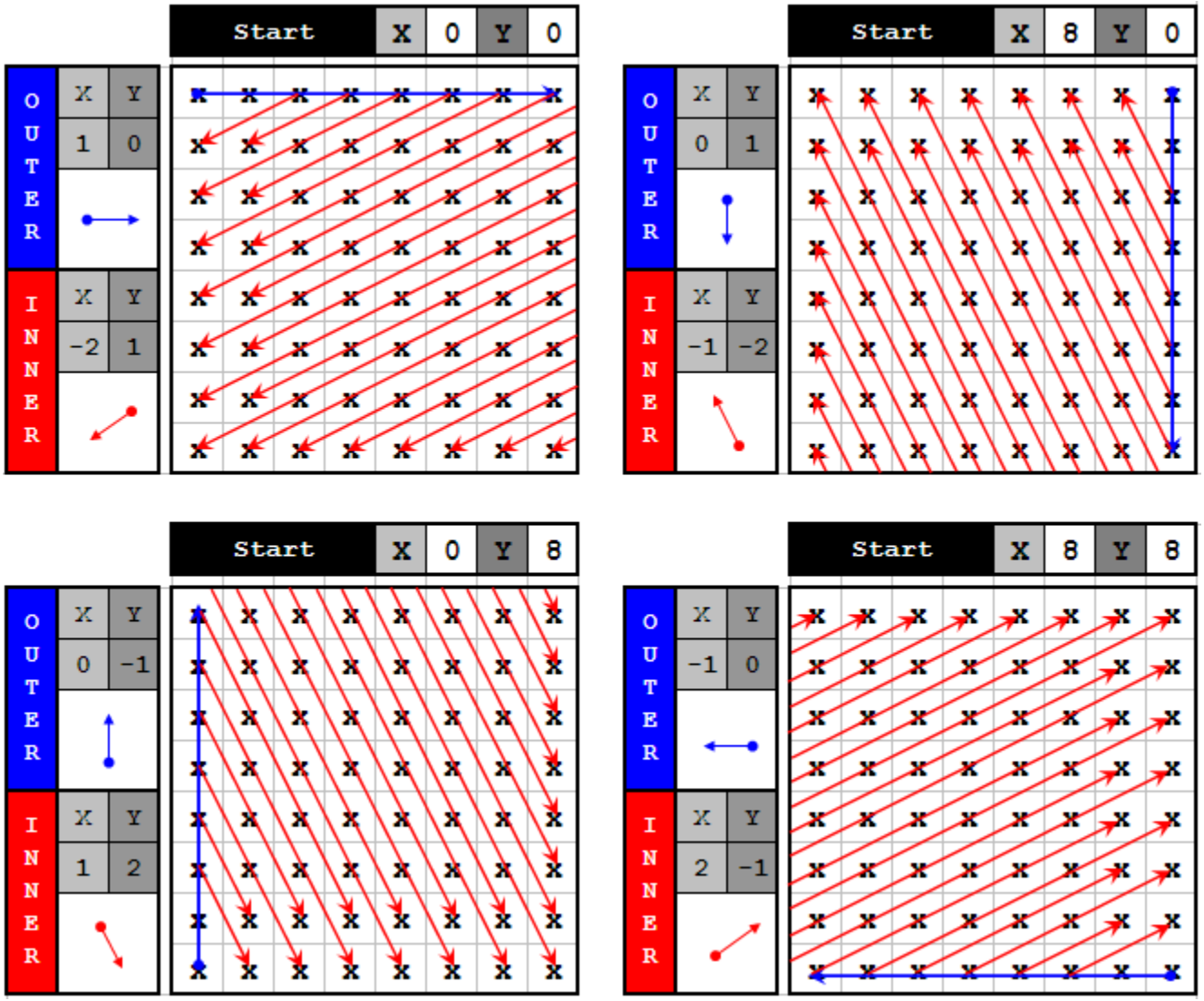
Basic Parameters for the Local Loop

The local inner and outer loop xy-pair parameters alone can describe a large variety of primitive walking patterns. Below are 9 primitive walking patterns generated by varying only the inner and outer unit step vectors of the local loop:



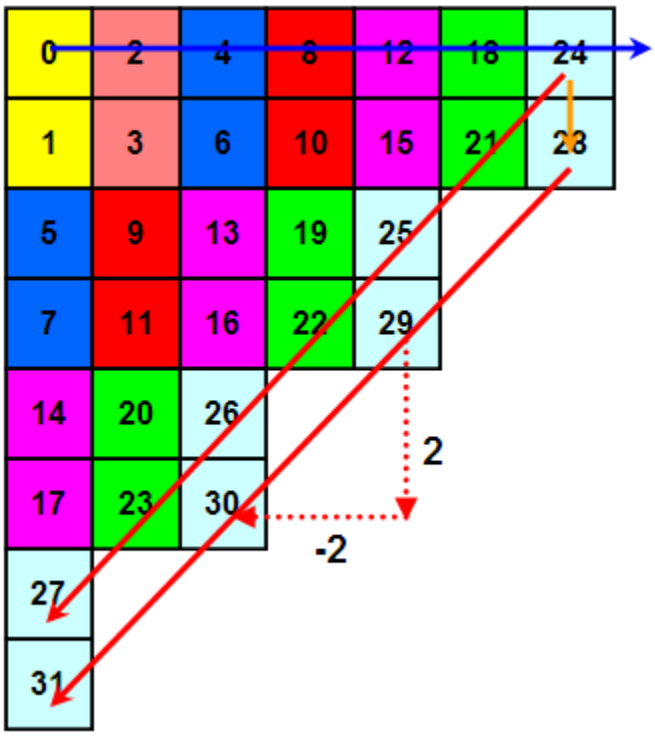
- The top row shows the outer unit vector pointing down (+Y) and the inner unit vector pointing right (+X). Rows and columns can easily be skipped by increasing the unit step vectors above one.
- The middle row the outer unit vector pointing right (+X) and the inner unit vector pointing down (+Y). Again, rows and columns are skipped by increasing the unit step vectors beyond one.
- The last row shows the capability to walk angles not perpendicular to the edge. The 1st shows a 45° walking pattern by setting the inner unit vector to (-1,1). The 2nd shows a checkerboard pattern by skipping every other outer loop and retaining the inner unit vector of (-1,1). The 3rd shows a 26.5° walking pattern by setting the inner unit vector to (-2,1).

The block resolution, shown as [8,8], and the starting location, currently [0,0], can be varied and the above patterns can be stretched and rotated many ways. The diagram below shows an example of where the start position and unit step vectors can be set to achieve a full rotation of the same pattern:



MbAff-Like Special Case in Local Loop

The local loop has an additional middle loop that is used to achieve some specific walking patterns, with MBAFF mode especially in mind. A pattern to handle MBAFF AVC content is to walk the top macroblocks of all macroblock pairs (MB-pairs) on a wavefront followed by the respective bottom macroblocks. The pattern is shown below.

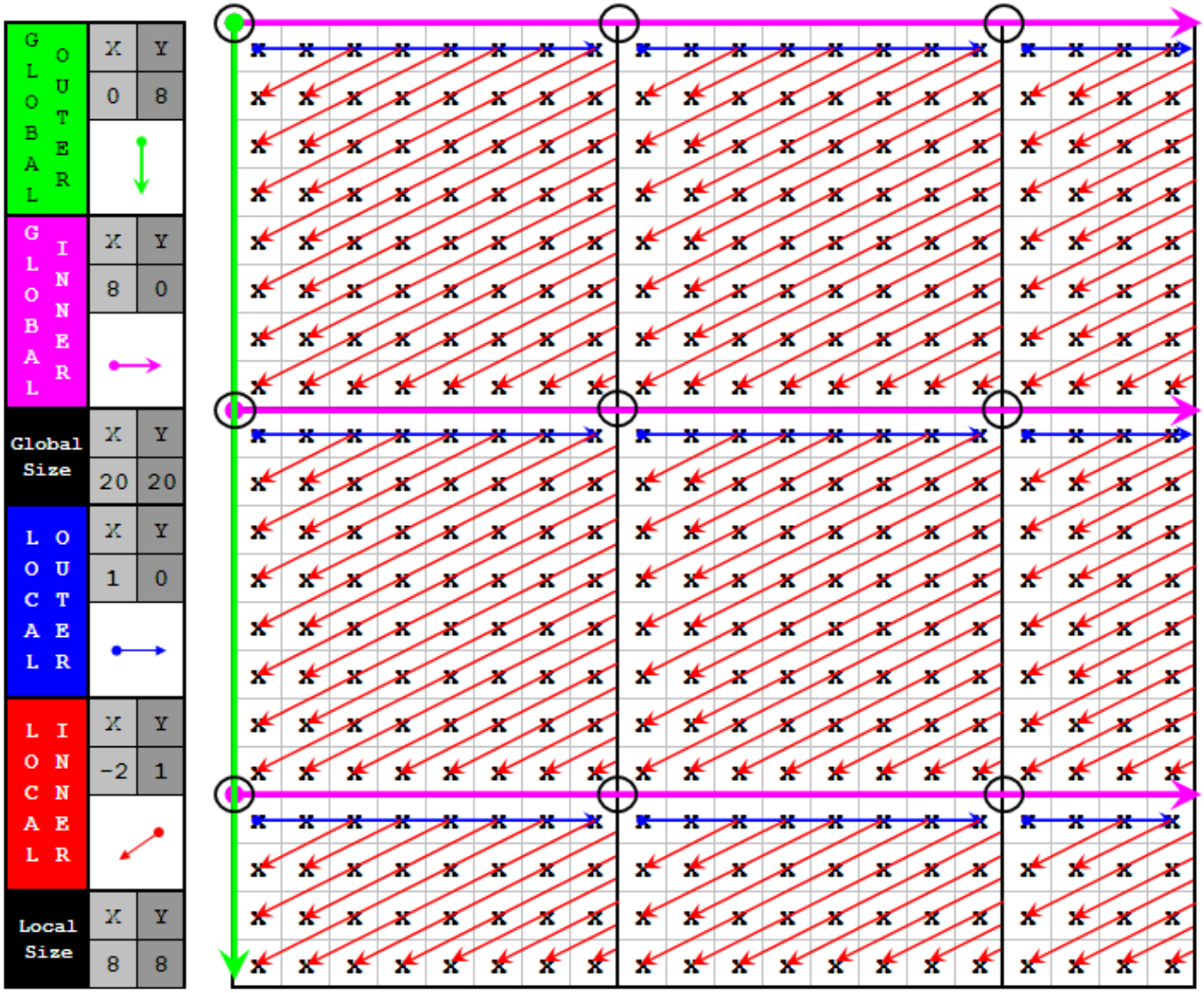


The outer loop unit step vector would be [1, 0] and the inner loop unit step vector would be [-2, 2]. A third loop is necessary to repeat the inner loop, only shifted down a unit before restarting. Thus, a middle loop with a unit step vector of [0,1] would achieve this MBAFF pattern. Additionally, the number of "extra steps" taken by the middle loop would be 1 in this case.

The addition of a middle loop also creates more overall flexibility, which seems necessary due to the integer-based unit step vector solution proposed (Manhattan distance issues etc.).

Global Loop

The same set of general parameters is used to describe the global loop as well. Thus, a global loop that is walking a raster-scan pattern can be combined with a local loop that is walking a 26.5° pattern (or vice-versa). As shown in the example below, if the local block size [8,8] is not an even multiple of the global resolution [20,20], the slack is still processed by dynamically changing the local block resolution.



The global loop will always resolve to be the upper-left corner of the local loop, shown above black circles. Note that local loop can still start in any corner of the local block, but the local (0,0) will always be the location where global loop begins the local loop, hence the upper-left corner.

The user can specify the starting location of the global loop as with the local loop. If the user were to set the global starting location to (16,16) in the previous example, after inverting the global outer and global inner unit step vectors the same pattern would be achieved in the reverse order. Note that the slack would still be handled along the right and bottom edge of the global image in that case. The user could have also started at (12,12) in which case the slack would be handled on the left and top faces.

Walker Algorithm Description

The walker algorithm has been tested and optimized in software. A high-level pseudo-code description is given below:

```
Walker(){ //C-Style Pseudo-Code of Walker Algorithm
    Load_Inputs_And_Initialize();
    While (Global_Outer_Loop_In_Bounds()){
        Global_Inner_Loop_Intialization();
        While (Global_Inner_Loop_In_Bounds()){
            Local_Block_Boundary_Adjustment();
            Local_Outer_Loop_Initialization();
            While (Local_Outer_Loop_In_Bounds()){
                Local_Middle_Loop_Initialization();
                While (Local_Middle_Steps_Remaining()){
                    Local_Inner_Loop_Initialization();
                    While (Local_Inner_Loop_Is_Shrinking()){
                        Execute();
                        Calculate_Next_Local_Inner_X_Y();
                    } //End Local Inner Loop
                    Calculate_Next_Local_Middle_X_Y();
                } //End Local Middle Loop
                Calculate_Next_Local_Outer_X_Y();
                Calculate_Next_Local_Inverse_Outer_X_Y();
            } //End Local Outer Loop
            Calculate_Next_Global_Inner_X_Y();
        } //End Global Inner Loop
        Calculate_Next_Global_Outer_X_Y();
    } //End Global Outer Loop
} //End Walker
```

The pseudo-code has the following characteristics:

- There are 5 levels of iteration
- The highest 2 levels are called “global” and the lowest 3 levels are called “local”

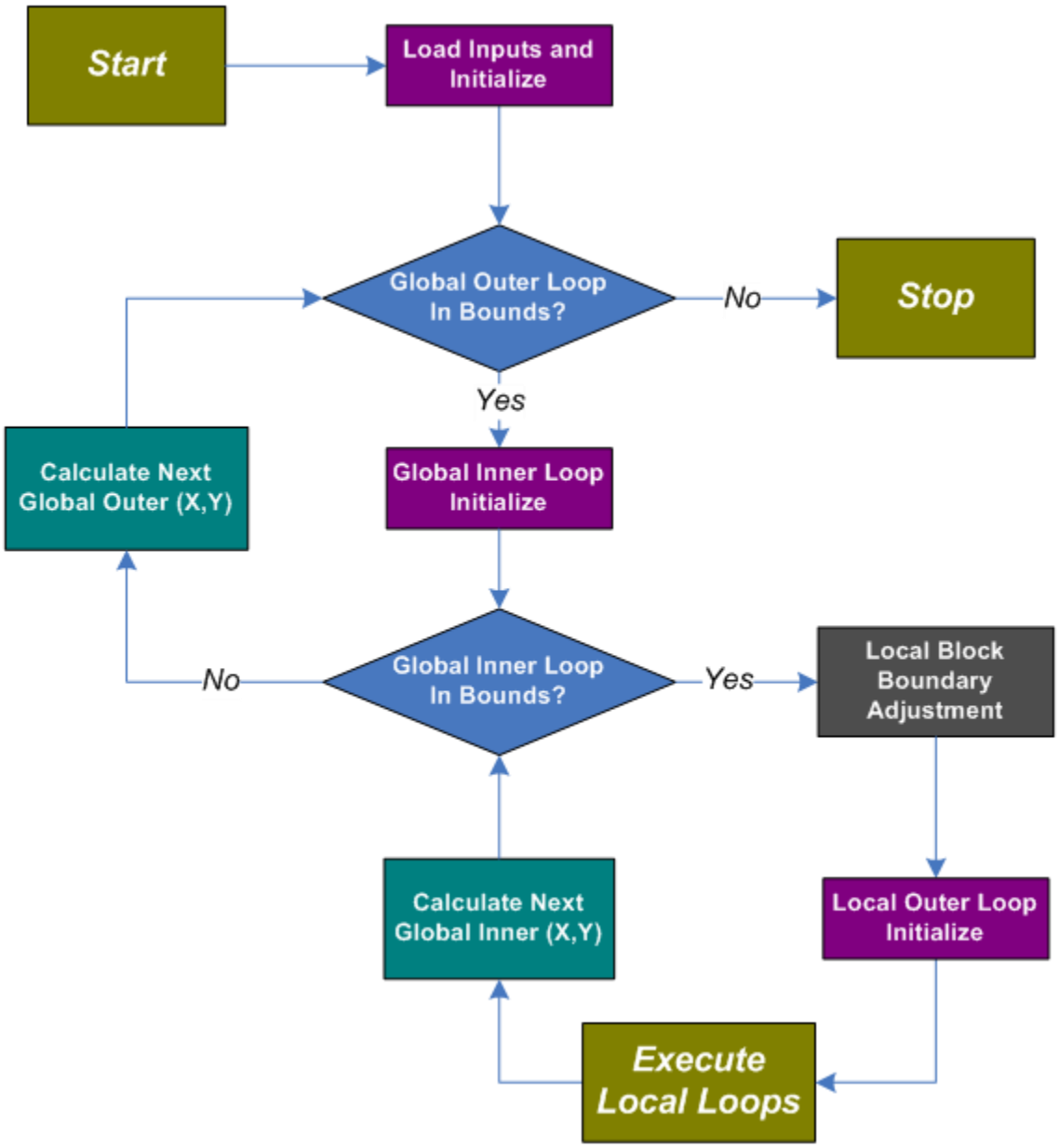
    - The global loop is split into an outer and an inner loop.
    - The local loop is split into an outer, a middle, and an inner loop.
    - A bounding box for the global and local resolution is defined by the user.
    - The starting location within each bounding box is also specified by the user.

- Each of the 5 loops has its own persistent
    - Current position (x,y)
    - Unit step vector (x,y)
- The final output (x,y) is a summation of the global x,y and the local x,y.
- The next (x,y) for given level can be calculated while the next lower level is still executing. Additionally, the result can be used to check to see if the current level will execute again once control is returned.

The flow of the global outer and inner loops is:

1. Check a bound condition
2. Initialize the next level loop
3. Execute the next level loop
4. When the next level loop fails its condition, calculate the next position for the current loop level and repeat.

Walker algorithm flowchart for the Global Loop



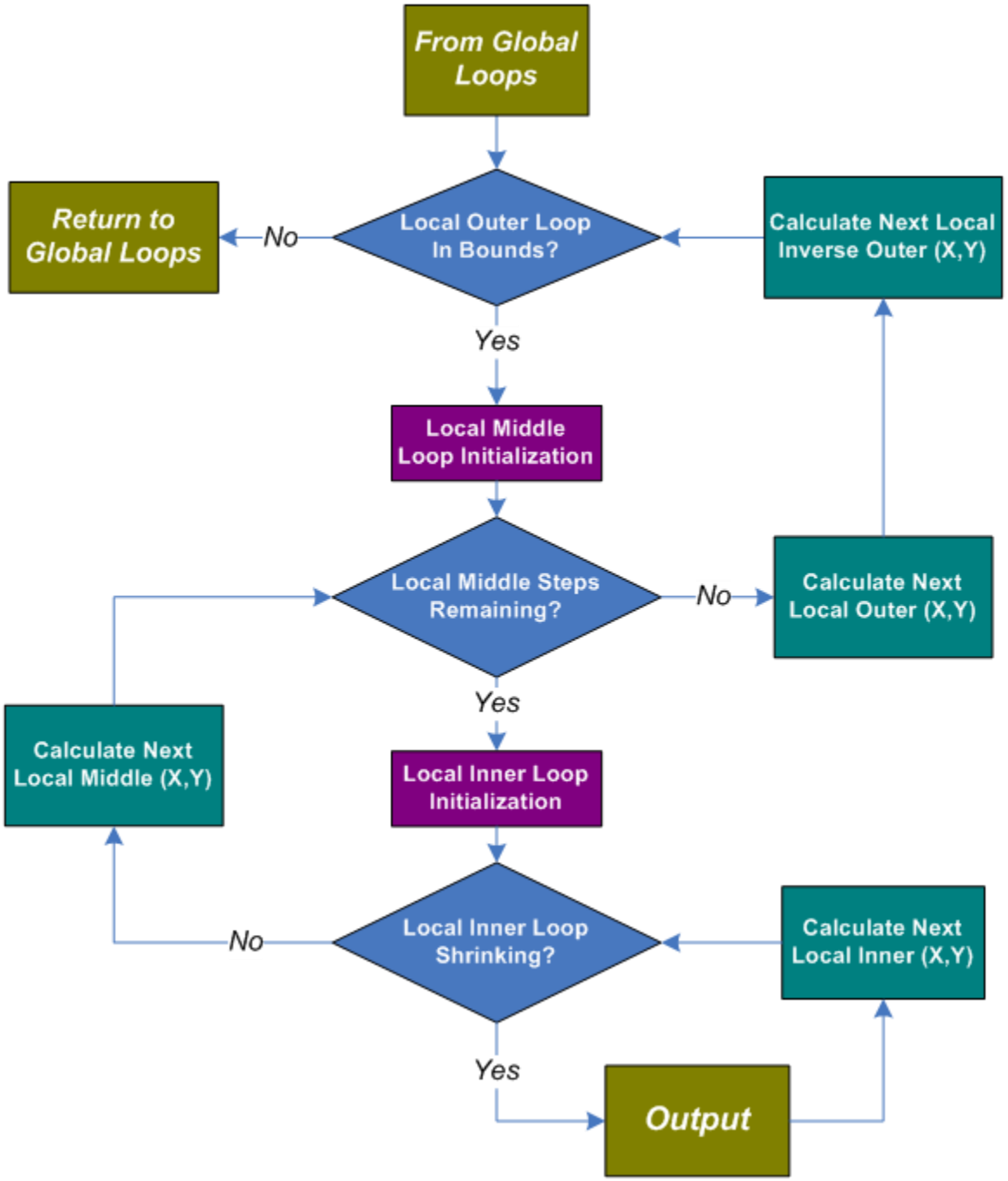
Take note of the grey box “Local Block Boundary Adjustment”. This logic is necessary to adjust the local block size when the distance between the current global position to the edge of the image is less than the local resolution. Additionally, the local starting positions might be modified here as well if the defined starting position is larger than the new local block size.

The flow of the 3 local loops does not vary much from the 2 global loops. The differences are:

- In addition to a boundary check, the local middle loop also ensures the number of middle steps is less than or equal to the user defined “number of extra steps”.

- The local inner loop only checks to see if the prior distance between the x,y starting and ending points are greater than their current distance. If this is true, it implies that the two inner loops are converging towards each other.
- When the middle loop check fails, both the starting points (local outer) and ending points (local inner) are updated.

Walker algorithm flowchart for the Local Loop



Barriers and Shared Local Memory

Barriers and Shared Local Memory can provide advantages to general media applications. Barriers can be used to synchronize between media threads more efficiently than using atomics, while SLM can be used to share data between tightly associated threads.

CHV, BSW allows Barriers and SLM to be used with a more generalized walker, as well as adding the ability to use the scoreboard at the same time. To implement this there needs to be an identifying number similar to the Thread Group ID which can be used to track and free resources.

For MEDIA_OBJECT_WALKER there are already numbers available which can be used as the group id – the various loop counts that the walker maintains. To provide flexibility, the programmer will be allowed to specify which loop counts will form the group id and which will count the threads inside each group. If cross-slice barriers are disabled (see section 1.7), the walker will be required to ensure that all threads in a group are dispatched to a single subslice so that the barrier and SLM are available to all group members. The programmer is responsible to ensure that the number of threads generated per group is not larger than the threads available in a subslice.

MEDIA_OBJECT_WALKER adds the X/Y values for the various loops added together to produce a single X/Y. Since walking patterns can be produced which have overlapping X/Y values, this can't be used for the global id, instead the execution counts (a count of how many times each loop is executed) are concatenated together to produce an id number. The GPGPU thread group id is a 96-bit number, so the 49 bits created by all the execution counts will fit easily.

The media walker has 5 nested loops for producing the X/Y; in addition there is an innermost color loop:

1. Color loop – 4 bits
2. Inner local loop execution count – 10 bits
3. Mid local loop execution count – 5 bits
4. Outer local loop execution count – 10 bits
5. Inner global loop execution count – 10 bits
6. Outer global loop execution count – 10 bits

The bits to use as the global id are specified with a parameter which specifies at which point to switch between the group id and the per thread id. Unused loops will always have execution counts of 0. The group id is formed from whichever execution counts are enabled in the Group ID Loop Select, with the selected execution counts concatenated into the LSBs of the group id, with any unused MSBs 0.

Example: if a MEDIA_OBJECT_WALKER specifies that the Outer local loop count and above will form the group id, then every iteration of the color, inner local, and mid local will dispatch a thread with the same group id. TSG will allocate a shared barrier and SLM (depending on what is specified in the Interface Descriptor) and ensure that all the threads go to the same subslice. The programmer ensures that this number of threads will fit on a single subslice, unless only global barriers are used, in which case the number of thread must fit on the system. The group id for this example is formed by cat(outer global exec count, inner global exec count, outer local exec count).

When the group id increments, the TSG will allocate a new barrier and SLM and pick an available subslice.

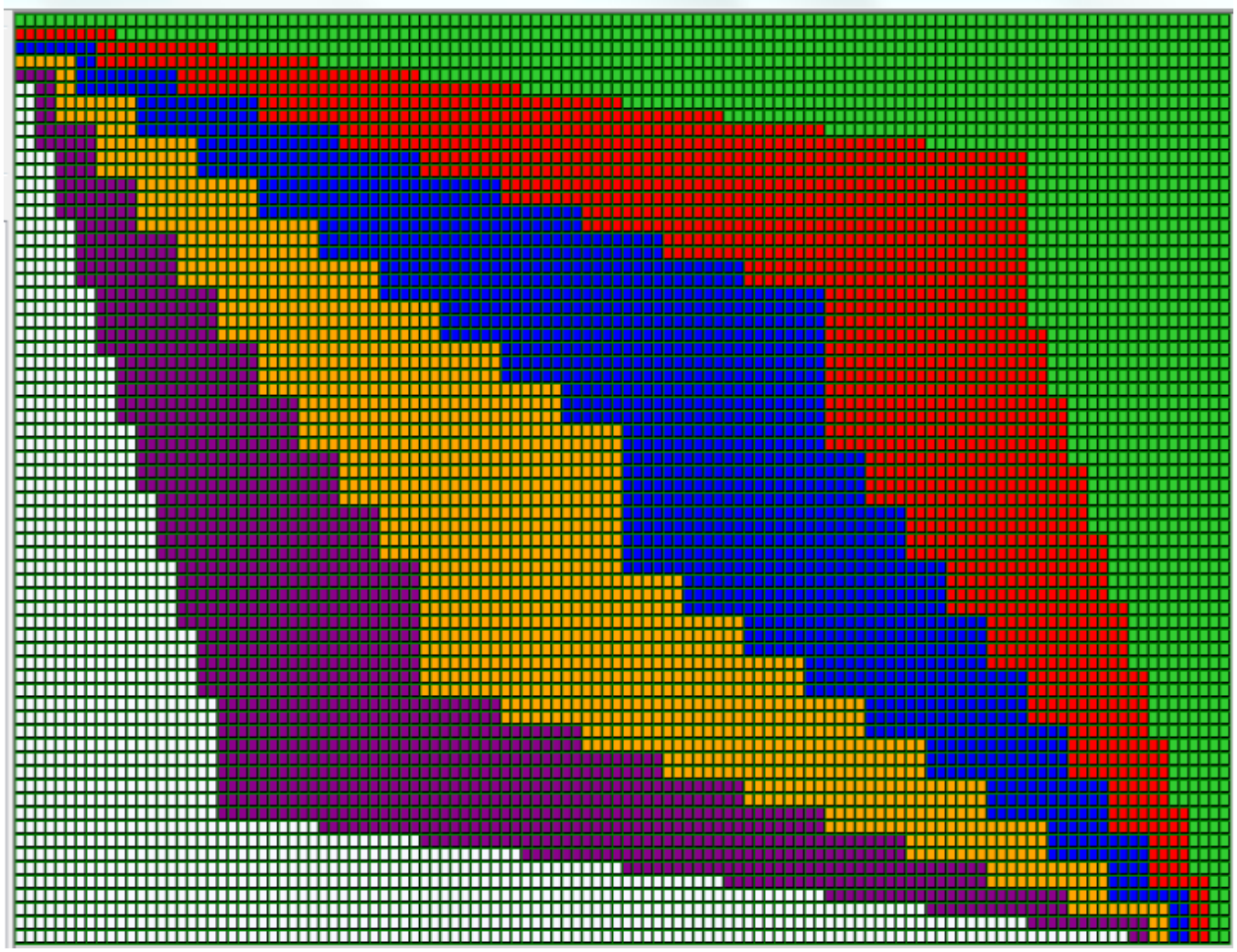
Before the MEDIA_OBJECT_WALKER command with groups there should be a MEDIA_STATE_FLUSH with a watermark bit and matching Interface Descriptor.

Flexible Dispatch of Local Loop

The local loop is automatically split between the availble subslices in such a way as to keep adjacent blocks next to each other to improve cache and execution efficency. The length of a single iteration of the local loop is split into equal segments, one for each subslice that is currently powered up.

The dispatches for the local loop are done such that one thread from each segment is dispatched before repeating the process: for example segment 0, thread 0 is followed by segment 1, thread 0 rather than segment 0, thread 1.

Example of splitting a wavefront between 6 destinations



Each color indicates a separate segment which is dispatched to a different subslice.

In the example above, each local loop walks a diagnal line from the lower left to the upper right, while the global loop steps between the lines. This is a typical usage model where the dependencies are to the left and above. The first few iterations are in the upper right corner and so have few blocks dispatched per local loop. Farther down, the length of the local loop gets large enough that the 6 available subslices are full of threads from each segment running in parallel and being dispatched in an even manner.

Scoreboard Control

A hardware mechanism controls the dispatch of root threads. Without using this hardware mechanism, only the dispatch of a SRT is managed by a parent root thread using the SRT message to TS.

There is a scoreboard hardware in TS unit. The scoreboard is addressed by the 18-bit (X, Y) scoreboard field in VFE DWord, where (X, Y) is typically used as the Cartesian coordinate of the working unit in a 2D frame but may be interpolated in other ways. When a root thread is dispatched, the entry at (X, Y) is marked. When the root thread is terminated, the corresponding bit in the scoreboard is cleared.

Each root thread may have up to eight dependencies. The dependency relation is described by the state value of Scoreboard Controls in terms of related distance of (deltaX, deltaY). There is a global scoreboard enabling in the state as well as the-per thread enabling for each dependency.

TS stalls the dispatch of a root thread if any scoreboard entry, which is denoted by (Scoreboard X + deltaX, Scoreboard Y + deltaY), matching with any enabled dependencies is marked as in-flight. The thread is dispatched only after all dependencies are cleared.

For a root thread, TS stalls the dispatch of the thread only if the dependent scoreboard entries of the thread are marked. It does not automatically stalls the dispatch for destination collision if (deltaX = 0, deltaY=0) is not set in the scoreboard state. This kind of scoreboard destination collision is due to the scoreboard wrap-around (or aliasing), which must be avoided. With 9-bit per X, Y field, the hardware scoreboard can support a frame that is subdivided up to 512x512 threads without a scoreboard aliasing.

In addition to the above 'stalling scoreboard', Media Pipe may also support a non-stalling scoreboard. With non-stalling, a thread is dispatched with the dependent threads marked. The thread dependency affects the issuing of a *sendc* instruction. See vol5d Execution Unit ISA for details.

Scoreboard Support in Device Hardware

Device	Stalling scoreboard	Non-Stalling scoreboard
[CHV, BSW]	Yes	Yes

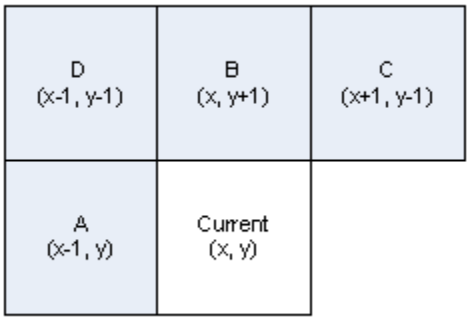
Programming Note	
Context:	Scoreboard Control
• The hardware scoreboard only handles root threads, but not child threads. This limitation may be revisited when future application requirement changes. • The usage of hardware scoreboard and SRT are mutually exclusive. In other words, when hardware scoreboard is used, SRT should not be issued.	

AVC-Style Dependency Example

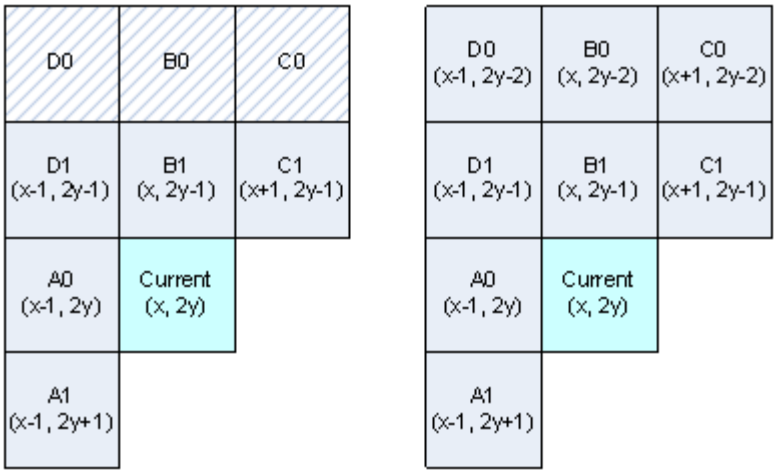
For AVD decoding, dependencies for a given macroblock may be set based on the availability of neighbor macroblocks, namely A, B, C, D and left-bottom neighbors (left-bottom only if MbAff = 1), as well as the current macroblock's address, MbAff flag and FieldMbFlag. For a macroblock in a progressive frame picture or a field picture, one macroblock may depend on up to four neighbors, A, B, C and D as shown in Neighbor addresses of a macroblock in a progressive frame picture (MbAff = 0) or a field picture with up to 4 dependencies. For a macroblock in a MbAff pair, it may depend on up to three, five or eight neighbors as shown in Neighbor addresses of the first macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies and Neighbor addresses of the second macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies, based on the current macroblock's address and FieldMbFlag.

The neighbor's availability depends on the corresponding **IntraPredAvailFlagA|B|C|D|E** flags for the macroblock (or the macroblock pair). Hardware assumes that the flags are set correctly in the MEDIA_OBJECT_EX command as shown in Macroblock indices for field picture destination. For simplicity, the left neighbor pair (A0 and A1) availability for a MbAff macroblock can be determined as a group by **IntraPredAvailFlagA | IntraPredAvailFlagE**. For the second macroblock in a 'frame' MbAff pair, it depends on the first macroblock in the pair and it is always available.

Neighbor addresses of a macroblock in a progressive frame picture (MbAff = 0) or a field picture with up to 4 dependencies



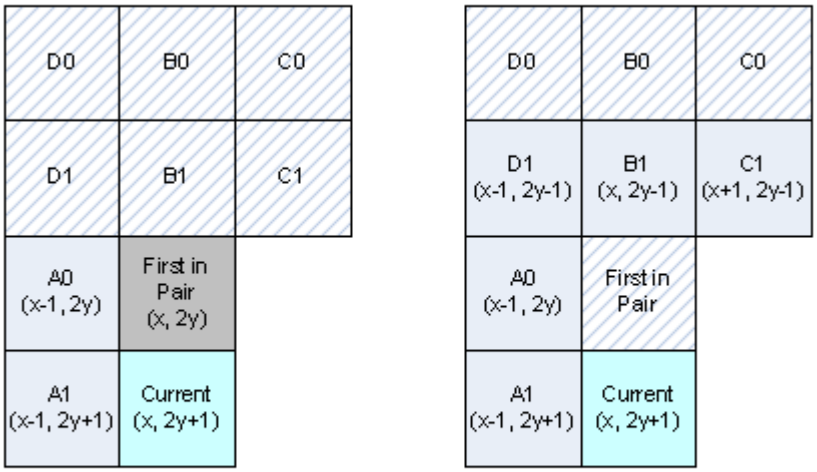
Neighbor addresses of the first macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies



(a) Neighbors for the first macroblock in a 'frame' MbAff pair

(b) Neighbors for the first macroblock in a 'field' MbAff pair

Neighbor addresses of the second macroblock in a MbAff frame picture (MbAff = 1) with up to 8 dependencies



(b) Neighbors for the second macroblock in a 'frame' MbAff pair

(b) Neighbors for the second macroblock in a 'field' MbAff pair

Neighbor Availability

MbAff	FieldMbFlag	VertOrigin[0]	A	B	C	D	LB	Description
0	0/1	0/1	□	□	□	□	•	Progressive or Field picture
1	0	0	□	□	□	□	□	1st Frame MbAff macroblock

MbAff	FieldMbFlag	VertOrigin[0]	A	B	C	D	LB	Description
1	0	1	□	na	0	na	□	2nd Frame MbAff macroblock
1	1	0	□	□	□	□	□	1st Field MbAff macroblock
1	1	1	□	□	□	□	□	2nd Field MbAff macroblock

Interface Descriptor Selection

In VLD mode, the Interface Descriptor Offset field in the MEDIA_OBJECT command is ignored by hardware. Instead, the interface descriptor offset is computed by hardware based on the decoded macroblock parameters and a remapping table.

First a macroblock index is computed based on parameters such as picture structure, motion type, prediction type, DCT type, intra-coding type and motion vector present information. *Interface Descriptor Selection* provides the macroblock index table for a frame-picture destination buffer (with Picture Structure = 11). *Interface Descriptor Selection* shows macroblock indices for a field-picture destination buffer (with Picture Structure = 01 or 10). As Picture Structure is a state variable that is not changed until a pipeline flush, the macroblock indices can be computed separately for different Picture Structure.

After the macroblock index is computed, it is used as the index into the Interface Descriptor Remap Table to derive the final interface descriptor offset value. The Interface Descriptor Remap Table is provided as part of the VLD state.

The interface descriptor offset value multiplied by the interface descriptor size is then added to the interface descriptor base pointer to generate the interface descriptor pointer for the post-VLD thread.

The last three columns in *Interface Descriptor Selection* and *Interface Descriptor Selection* indicate whether a macroblock index is applicable for a given Picture Coding type (I, P or B). A 'Y' (or a 'N') means the macroblock index on the row is valid (or invalid) for the Picture Type shown on the column. Taking a frame picture destination for example, only macroblock indices 0 and 8 are valid for an I-picture; indices 0-3 and 8-11 are valid for a P-picture; and for a B-picture, only indices 3 and 11 are not valid.

Developers can use the remap table for kernel development to fine-tune system performance and reduce software complexity. For example, if the destination is a frame picture, the kernel for a macroblock with dual-prime motion in a P-picture (macroblock index = 3) may be identical to that for a macroblock with bidirection field motion in a B picture (macroblock index = 7). A common set of interface descriptors can be configured once for frame picture destinations, and reused without change when the destination is of I-, P- and B- picture coding type.

In another case, if it is determined that kernel software is responsible for handling DCT types for a frame picture destination, then macroblock index i and i+8 , for i = 0 to 7, can be mapped to the same interface descriptor.

Macroblock Indices for Frame Picture Destination

Macroblock Index	Interface Descriptor Kernel Function (Frame Picture Destination)	I	P	B
0	I macroblock	Y	Y	Y
1	Forward frame motion	N	Y	Y
2	Forward field motion	N	Y	Y
3	P picture, dual-prime motion	N	Y	N
4	Backward frame motion	N	N	Y
5	Backward field motion	N	N	Y
6	Bidirectional frame motion	N	N	Y
7	Bidirectional field motion	N	N	Y
8	I macroblock with field DCT	Y	Y	Y
9	Forward frame motion with field DCT	N	Y	Y
10	Forward field motion with field DCT	N	Y	Y
11	P picture, dual-prime motion with field DCT	N	Y	N
12	Backward frame motion with field DCT	N	N	Y
13	Backward field motion with field DCT	N	N	Y
14	Bidirectional frame motion with field DCT	N	N	Y
15	Bidirectional field motion with field DCT	N	N	Y

Macroblock Indices for Field Picture Destination

Macroblock Index	Interface Descriptor Kernel Function (Field Picture Destination)	I	P	B
0	I macroblock	Y	Y	Y
1	Forward field motion	N	Y	Y
2	Forward 16x8 motion	N	Y	Y
3	P picture, dual-prime motion	N	Y	N
4	Backward field motion	N	N	Y
5	Backward 16x8 motion	N	N	Y
6	Bidirectional field motion	N	N	Y
7	Bidirectional 16x8 motion	N	N	Y

VC1-Style Dependency Example

For VC1, only one dependency may be set depending on the availability of the upper neighbor macroblock.

Macroblock sequence order in a VC-1 picture with WidthInMblk = 5 and HeightInMblk = 6

	0	**1**	**2**	**3**	**4**
0	0	1	2	3	4
1	5	6	7	8	9
2	10	11	12	13	14
3	15	16	17	18	19
4	20	21	22	23	24
5	25	26	27	28	29

Multiple Slice Considerations

For products with multiple slices, the Render Cache is separate per slice with no hardware coherency. This means that the programmer must ensure coherency by one of these methods:

- Using write commit when writing to the Render Cache.
- Using Data Cache instead of the Render Cache.
- Different slices only access separate cache lines. using a hashing algorithm combined with the slice select bits of the MEDIA_OBJECT/GPGPU_OBJECT commands.

Interrupt Latency

Command Streamer is capable of context switching between primitive commands.

For all independent threads, it is not much a problem. The interrupt latency is dictated by the longest command that is likely to have the largest number of threads. For VLD mode, such a command may be corresponding to a largest slice in a high definition video frame. This is application dependent, there are not much host software can do. For Generic mode, programmer should consider to constrain the compute workload size of each thread.

In modes with child threads, a root thread may persist in the system for long period of time – staying until its child threads are all created and terminated. Therefore, the corresponding primitive command may also last for long time. The Software designer should partition the workload to restrict the duration

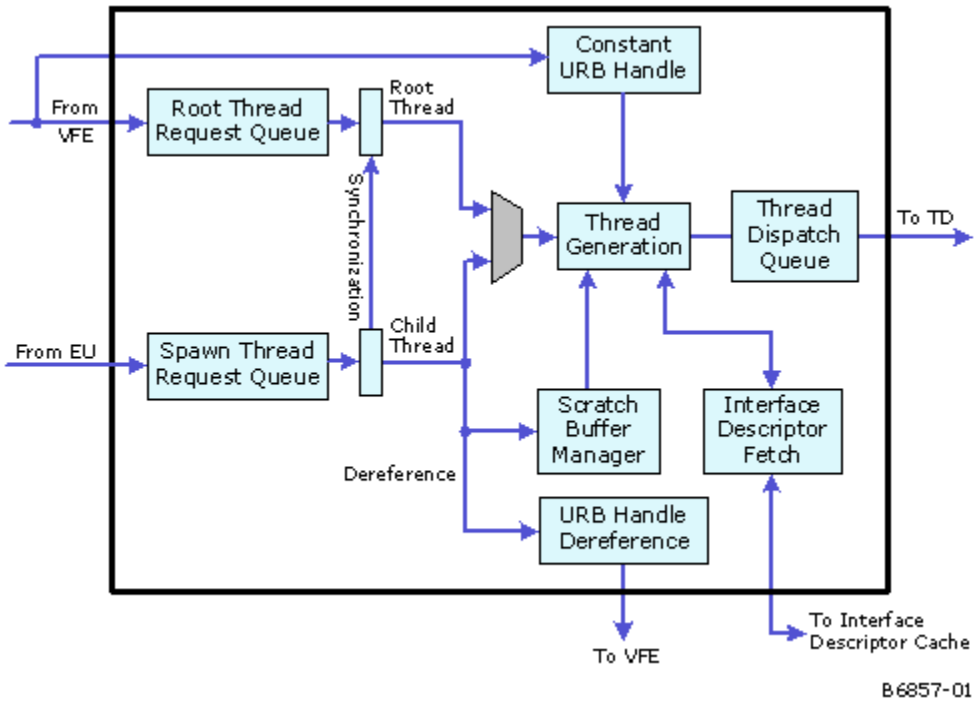
of each root thread. For example, this may be achieved by partitioning a video frame and assigning separate primitive commands for different data partitions.

In modes with synchronized root threads, a synchronized root thread is dependent on a previous root or child thread. This means context switch is not allowed between the primitive command for the synchronized root thread and the one for the depending thread. So no command queue arbitration should be allowed between them. Software designer should also restrict the duration of such non-interruptible primitive command segments.

Thread Spawner Unit

The Thread Spawner (TS) unit is responsible for making thread requests (root and child) to the Thread Dispatcher, managing scratch memory, maintaining outstanding root thread counts, and monitoring the termination of threads.

Thread Spawner block diagram



Root Threads and Child Threads

Thread requests sourced from VFE are called **root threads**. These threads may be creating subsequent child threads.

Root Threads

A root thread may be a macroblock thread created by VFE as in VLD mode, or may be a general-purpose thread assembled by VFE according to full description provided by host software in Generic mode. Thread requests are stored in the Root Thread Queue. TS keeps everything needed to get the root threads ready for dispatch and then tracks dispatched threads until their retirement.

TS arbitrates between root thread and child thread. The root thread request queue is in the arbitration only if the number of outstanding threads does not exceed the maximum root thread state variable. Otherwise, the root thread request queue is stalled until some other root threads retire/terminate.

Once a root thread is selected to be dispatched, its lifecycle can be described by the following steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.
    - Once TS receives the interface descriptor, it checks whether maximum concurrent root thread number has reached to determine whether to make a thread dispatch request or to stall the request until some other root threads retire. If the thread requests the use of scratch memory, it also generates a pointer into the scratch space.
2. TS then builds the transparent header and the R0 header.
3. Finally, TS makes a thread request to the Thread Dispatcher.
4. TS keeps track of dispatched thread, and monitors messages from the thread (resource dereference and/or thread termination). When it receives a root thread termination message, it can recover the scratch space and thread slot allocated to it. The URB handle may also be dereferenced for a terminated root thread for future reuse. It should be noted that URB handle dereference may occur before a root thread terminates. See detailed description in the Media Message section.
    - It is the root thread's responsibility (software) to guarantee that all its children have retired before the root thread can retire.

URB Handles

VFE is in charge of allocating URB handles for root threads. One URB handle is assigned to each root thread. The handle is used for the payload into the root thread.

Children Present is a command variable in the _OBJECT command.

If Children Present is not set (root-without-child case), TS signals VFE to dereference the URB handle immediately after it receives acknowledgement from TD that the thread is dispatched.

If Children Present is set (root-with-child case), the URB handle is forwarded to the root thread and serves as the return URB handle for the root thread. TS does not signal deference at the time of dispatch. TS signals URB handle deference only when it receives a resource dereference message from the thread.

Root to Child Responsibilities

Any thread created by another thread running in an EU is called a **child thread**. Child threads can create additional threads, all under the tree of a root which was requested via the VFE path.

A root thread is responsible of managing pre-allocated resources such as URB space and scratch space for its direct and indirect child threads. For example, a root thread may split its URB space into sections. It can use one section for delivering payload to one child thread as well as forwarding the section to the child thread to be used as return URB space. The child thread may further subdivide the URB section into subsections and use these subsections for its own child threads. Such process may be iterated. Similarly, a root thread may split its scratch memory space into sections and give one scratch section for one child thread.

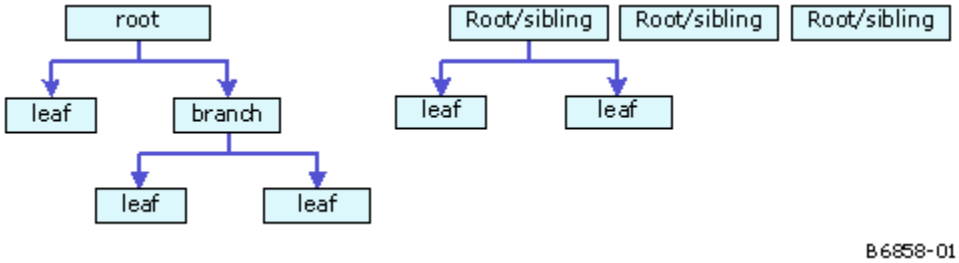
TS unit only enforces limitation on number of outstanding root threads. It is the root threads' responsibility to limit the number of child threads in their respected trees to balance performance and avoid deadlock.

Multiple Simultaneous Roots

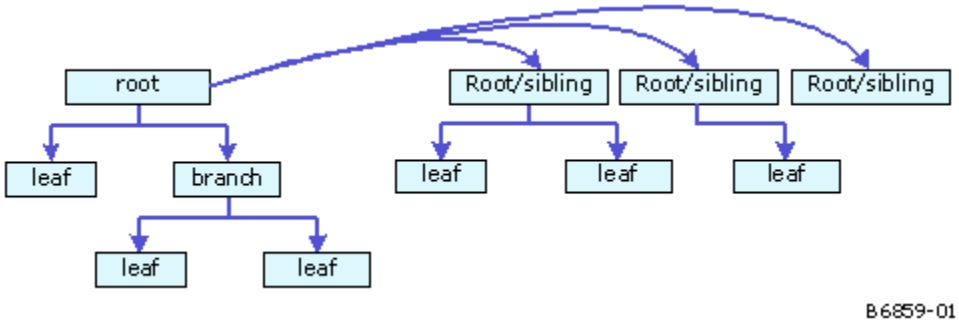
Multiple root threads are allowed concurrently running in GEN4 execution units. As there is only one scratch space state variable shared for all root threads, all concurrent root thread requiring scratch space share the same scratch memory size. *Multiple Simultaneous Roots* depicts two examples of thread-thread relationship. The left graph shows one single tree structure. This tree starts with a single root thread that generates many child threads. Some child threads may create subsequent child threads. The right graph shows a case with multiple disconnected trees. It has multiple root threads, showing sibling roots of disconnected trees. Some roots may have child threads (branches and leafs) and some may not.

There is another case (as shown in *Multiple Simultaneous Roots*) where multiple trees may be connected. If a root is a synchronized root thread, it may be dependent on a preceding sibling root thread or on a child thread.

Examples of thread relationship



A example of thread relationship with root sibling dependency



Synchronized Root Threads

A synchronized root thread (SRT) originates from a MEDIA_OBJECT command with Thread Synchronization field set. Synchronized root threads share the same root thread request queue with the non-synchronized roots. A SRT is not automatically dispatched. Instead, it stays in the root thread request queue until a spawn-root message is at the head of the child thread request queue. Conversely, a spawn-root message in the child thread request queue will block the child thread request queue until the head of root thread request queue is a SRT. When they are both at the head of queues, they are taken out from the queue at the same time.

A spawn-root message may be issued by a root thread or a child thread. There is no restriction. However, the number of spawn-root messages and the number of SRT must be identical between state changes. Otherwise, there can be a deadlock. Furthermore, as both requests are blocking, synchronized root threads must be used carefully to avoid deadlock.

When Scoreboard Control is enabled, the dispatch of a SRT originated from a MEDIA_OBJECT_EX command is still managed by the same way in addition to the hardware scoreboard control.

Deadlock Prevention

Root threads must control deadlock within their own child set. Each root is given a set of preallocated URB space; to prevent deadlock it must make sure that all the URB space is not allocated to intermediate children who must create more children before they can exit.

There are limits to the number of concurrent threads. The upper bound is determined by the number of execution units and the number of threads per EU. The actual upper bound on number of concurrent

threads may be smaller if the GRF requirement is large. Deadlock may occur if a root or intermediate parent cannot exit until it has started its children but there is no space (for example, available thread slot in execution units) for its children to start.

To prevent deadlock, the maximum number of root threads is provided in VFE state. The Thread Spawner keeps track of how many roots have been spawned and prevents new roots if the maximum has been reached. When child threads are present, it is software's responsibility to constrain child thread generation, particularly the generation of child threads that may also spawn more child threads.

Child thread dispatch queue in TS is another resource that needs to be considered in preventing deadlock. The child thread dispatch queue in TS is used for (1) message to spawn a child thread, (2) message to spawn a synchronized root thread, and (3) thread termination message. If this queue is full, it will prevent any thread to terminate, causing deadlock.

For example, if an application only has one root thread (max # of root threads is programmed to be one). This root thread spawns child threads. In order to avoid deadlock, the maximum number of outstanding child thread that this root thread can spawn is the sum of the maximum available thread slots plus the depth of the child thread dispatch queue minus one.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1)

Adding other root threads (synchronized and/or non-synchronized) to the above example, the situation is more complicated. A conservative measure may have to use to prevent deadlock. For example, the root thread spawning child threads may have to exclude the max number of root threads as in the following equation to compute the maximum number of outstanding child threads to be dispatched.

Max_Outstanding_Child_Threads = (Thread Slot Number – 1) + (TS Child Queue Depth – 1) – (Max Root Threads-1)

Child Thread Life Cycle

When a (parent) thread creates a child thread, the parent thread behaves like a fixed function. It provides all necessary information to start the child thread, by assembling the payload in URB (including R0 header) and then sending a spawn thread message to TS with following data:

- An interface descriptor pointer for the child thread.
- A pointer for URB data

The interface descriptor for a child may be different from the parent – how the parent determines the child interface descriptor is up to the parent, but it must be one from the interface descriptor array on the same interface descriptor base address.

The URB pointer is not the same as a URB handle. It does not have an URB handle number and does not appear in any handle table. This is acceptable because the URB space is never reclaimed by TS after a child is dispatched, but rather when the parent releases its original handles and/or retires.

Programming Note	
Context:	Child Thread Life Cycle
The child request is stored in the child thread queue. The depth of the queue is limited to 8, overrun is prevented by the message bus arbiter which controls the message bus. The arbiter knows the depth of the queue and will only allow 8 requests to be outstanding until the TS signals an entry has been removed.	

As mentioned previously, child threads have higher priority over root threads. Once TS selects a child thread to dispatch, it follows these steps:

1. TS forwards the interface descriptor pointer to the L1 interface descriptor cache (a small fully associated cache containing up to 4 interface descriptors). The interface descriptor is either found in the cache or a corresponding request is forwarded to the L2 cache. Interface descriptors return back to TS in requesting order.
2. TS then builds the transparent header but not the R0 header.
3. Finally, TS makes a thread request to the Thread Dispatcher.
4. Once the dispatch is done, TS can forget the child – unlike roots, no bookkeeping is done that has to be updated when the child retires.

If more data needs to be transferred between a parent thread and its child thread than that can fit in a single URB payload, extra data must be communicated via shared memory through data port.

Arbitration between Root and Child Threads

When both root thread queue and child thread queue are both non-empty, TS serves the child thread queue. In other words, child threads have higher priority over root threads. The only condition that the child thread queue is stalled by the root thread queue is that the head of child thread queue is a root-synchronization message and the head of root thread queue is not a synchronized root thread.

Persistent Root Thread

A persistent root thread in general stays in the system for a long period of time. It is normally a parent thread, and only one PRT is allowed in the system at a time.

On a context switch interrupt, instead of proceeding to completion, a PRT can save its software context and terminate. The PRT can be restarted later, even if it had completed normally the last time it was executed. Therefore, the PRT must always save enough context (via data port messages to a predefined surface) to allow it to restart from where it left off (including determining that it has nothing left to do).

Because only one PRT can execute at a time, once the next PRT starts, the previous one will never be restarted, thus the context save surface can be reused from one PRT to the next.

A PRT may check the Thread Restart Enable bit in the R0 header to find out whether it is a fresh start or resumed from a previous interrupt and then can continue operations from that previously saved context.

A PRT can be interleaved with other root (such as parent root thread, or synchronized root thread) and child threads. A parent root thread is not necessarily a PRT, and doesn't have to be as long as it can be finished in deterministic time that is shorter than required for fine-grain context switch interrupt.

Use of PRT must follow the following rule:

- There can only be one PRT in the media pipeline at a given time. That means, there shall not be any other media primitive commands (MEDIA_OBJECT or MEDIA_OBJECT_EX) between it and the previous MI_FLUSH command. In other words, when multiple such PRTs are used in a sequence of media primitive commands, MI_FLUSH must be inserted.

Media State Model

The media state model is based on in-line state load mechanism. VFE state, URB configuration and Interface Descriptors are loaded to VFE hardware through state commands.

All Interface Descriptors have the same size and are organized as a contiguous array in memory. They can be selected by Interface Descriptor Index for a given kernel. This allows different kinds of kernels to coexist in the system.

Pipeline (Media) Bits[28:27]	Opcode Bits[26:24]	Sub Opcode Bits[23:16]	Command
2h	0h	00h	MEDIA_VFE_STATE
2h	0h	01h	MEDIA_CURBE_LOAD
2h	0h	02h	MEDIA_INTERFACE_DESCRIPTOR_LOAD

Media State and Primitive Commands

This section contains various commands for media, all with the RenderCS source.

MEDIA_VFE_STATE

MEDIA_CURBE_LOAD

MEDIA_INTERFACE_DESCRIPTOR_LOAD

Interface Descriptor Data payload as pointed to by the Interface Descriptor Data Start Address:

INTERFACE_DESCRIPTOR_DATA

Programming Restriction: Back to back interface descriptor load commands, must have a VFE State command inserted between them.

MEDIA_STATE_FLUSH

The MEDIA_OBJECT command is the basic media primitive command for the media pipeline. It supports loading of inline data as well as indirect data. At least one form of payload (either inline, indirect, or CURBE) must be sent with the MEDIA_OBJECT.

MEDIA_OBJECT

MEDIA_OBJECT_PRT

MEDIA_OBJECT_GRPID

The MEDIA_OBJECT_WALKER command uses the hardware walker in VFE for generating threads associated with a rectangular shaped object. It only supports loading of inline data or CURBE but not indirect data. At least one form of payload must be sent. Control of scoreboards (up to 8) is implicit based on the (X, Y) address of the generated thread and the scoreboard control state.

The command can be used only in Generic modes.

When **Use Scoreboard** field is set, the (X, Y) address and the Color field of the generated thread are used in the hardware scoreboard and the thread dependencies are set by states from the MEDIA_VFE_STATE command.

One or more threads may be generated by this command. This command does not support indirect object load. When inline data is present, it is repeated for all threads it generates. Unlike CURBE, which requires pipeline flush for change, continued change of this kind of 'global' (in the sense of shared by multiple threads from this command) data is supported when MEDIA_OBJECT_WALKER commands are issued without a pipeline flush in between.

Media State and Primitive Command Workarounds

Media State and Primitive Commands have some subtle programming restrictions and workarounds, as listed below.

Programming Note
The MEDIA_STATE_FLUSH command is updated to optionally specify all the resources required for the next thread group via an interface descriptor – if the resources are not available the group cannot start.

Media Messages

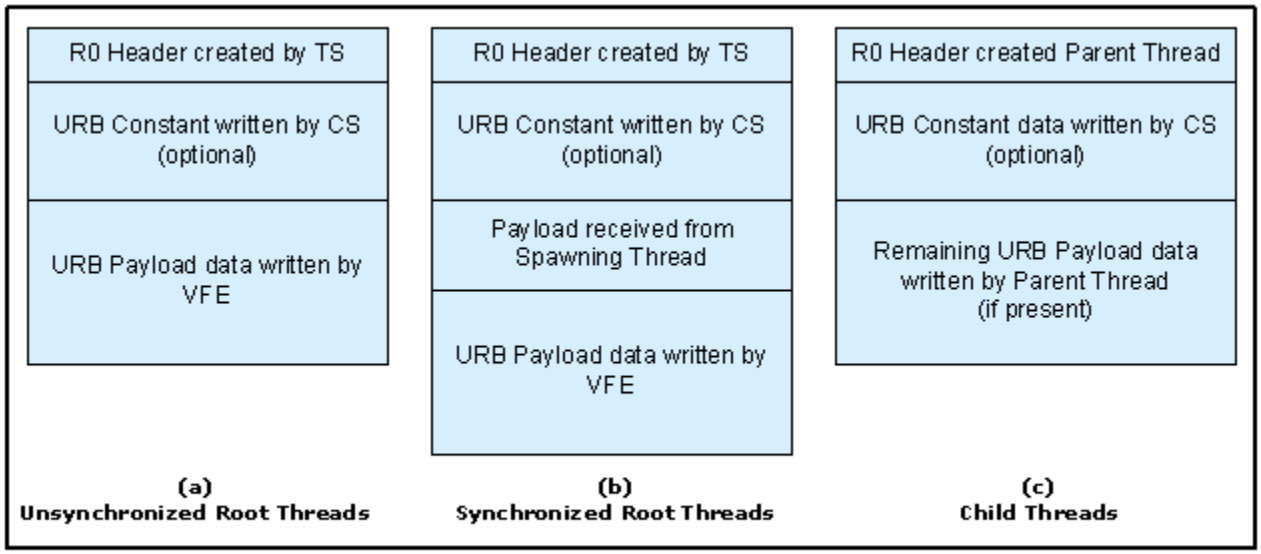
All message formats are given in terms of dwords (32 bits) using the following conventions:

- Dispatch Messages: **R**p.d
- SEND Instruction Messages: **M**p.d

Thread Payload Messages

The root thread's register contents differ from that of child threads, as shown in *Thread Payload Messages*. The register contents for a synchronized root thread (also referred to as 'spawned root thread') and an unsynchronized one are also different. Whether the URB Constant data field is present or not is determined by the interface descriptor of a given thread. This applies to both root and child threads. When URB Constant data field is present for a synchronized root thread, URB constant data field is before the data field received from the spawning thread, which is also before the URB payload data.

Thread payload message formats for root and child threads



B6863-01

Generic Mode Root Thread

The following table shows the R0 register contents for a Generic mode root thread, which is generated by TS. The remaining payloads are application dependent.

R0 Header of a Generic Mode Root Thread

DWord	Bits	Description
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9	Reserved: MBZ
	8:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion. Format = U9
R0.4	31:5	**Binding Table Pointer.** The 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** The amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2	31:28	Reserved: MBZ
	27:16	Reserved: MBZ
	15:10	Reserved: MBZ
	9:4	**Interface Descriptor Offset.** The offset from the interface descriptor base pointer to the interface descriptor that applies to this object, in units of interface descriptors. Format = U6
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX): This field specifies which dependency color the

DWord	Bits	Description
		current thread belongs to. It affects the dependency scoreboard control. Format = U4
R0.1	31:28	Reserved: MBZ
	26:25	Reserved: MBZ
	24:16	**Scoreboard Y** This field provides the Y term of the scoreboard value of the current thread. Format = U9
	15:11	Reserved: MBZ
	10:9	Reserved: MBZ
	8:0	**Scoreboard X** This field provides the X term of the scoreboard value of the current thread. Format = U9
R0.0	31:24	**Scoreboard Mask.** Each bit indicates the corresponding dependency scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. **Bit n (for n = 0...7):** Scoreboard n is dependent, where bit 24 maps to n = 0. Format = TRUE/FALSE
	23:16	Reserved: MBZ
	15:0	**URB Handle.** This is the URB handle indicating the URB space for use by the root thread and its children.

Root Thread from MEDIA_OBJECT_PRT

The root thread payload message for a MEDIA_OBJECT_PRT command has a fixed format independent of the VFE mode (e.g. Generic mode or AVC-IT mode). One example GRF register location is given for the condition that CURBE is disabled.

Root Thread Payload Layout for a MEDIA_OBJECT_PRT Command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	N/A	**Constants from CURBE when CURBE is enabled** m is a non-negative value.
R(m+1)	R1	**In-line Data block.**

The R0 header field is as the following, which is the same as in other modes except the Thread Restart Enable bit (bit 0 of R0.2).

R0 Header of the Thread Payload of a MEDIA_OBJECT_PRT Command

DWord	Bit	Description
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1K-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9:8	Reserved: MBZ
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion.
R0.4	31:5	**Binding Table Pointer:** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power that two is raised to, to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2	31:4	**Interface Descriptor Pointer.** Specifies the 16-byte aligned pointer to *this thread's* interface descriptor. Can be used as a base from which to offset child thread's interface descriptor pointers. Format = GeneralStateOffset[31:4]
	3:1	Reserved: MBZ
	0	**Thread Restart Enable.** If set, indicates that the persistent root thread (PRT) is being restarted, and context should be restored from the context save area before executing. Format = Enable
R0.1	31:0	Reserved: MBZ
R0.0	31:16	Reserved: MBZ
	15:0	**URB Handle.** This is the URB handle indicating the URB space for use by the root thread and its children.

The inline data block field is the same as in the MEDIA_OBJECT_EX command with zero-filled partial GRF.

Root Thread from MEDIA_OBJECT_WALKER

The root thread payload message for an MEDIA_OBJECT_WALKER command, which must be in Generic mode, has the same format as that of the generic mode root thread format.

Root thread payload layout for a MEDIA_OBJECT_WALKER command

GRF Register	Example	Description
R0	R0	**R0 header**
R1 – R(m)	n/a	**Constants from CURBE when CURBE is enabled** m is a non-negative value
R(m+1)	R1	**In-line Data block**.

The R0 header field is identical to that of Generic Mode Root Thread.

The inline data block field is the same as in the MEDIA_OBJECT command with zero-filled partial GRF.

There is no indirect data block field.

MEDIA_OBJECT_GRPID and MEDIA_OBJECT_WALKER with Groups Payload

The RO header of the MEDIA_OBJECT_GRPID & MEDIA_OBJECT_WALKER Payload with groups enabled:

DWord	Bits	Description
R0.7	31:0	**Group ID LSB.** This is the LSBs of the Group ID. For MEDIA_OBJECT_GRPID threads this is the entire group id. For MEDIA_OBJECT_WALKER threads the interpretation depends on the Group ID Loop Select: 0: No groups, field is 0. 1: cat(InnerGlobalCnt[6:0], OuterLocalCnt[9:0], MidLocalCnt[4:0], InnerLocalCnt[9:0]); rest of group id is in R0.2. 2: cat(OuterGlobalCnt[6:0], InnerGlobalCnt[9:0], OuterLocalCnt[9:0], MidLocalCnt[4:0]); rest of group id is in R0.2. 3: cat(2'b0, OuterGlobalCnt[9:0], InnerGlobalCnt[9:0], OuterLocalCnt[9:0]). 4: cat(12'b0, OuterGlobalCnt[9:0], InnerGlobalCnt[9:0]). 5: cat(22'b0, OuterGlobalCnt[9:0]).
R0.5	31:10	**Scratch Space Pointer.** Specifies the 1k-byte aligned pointer to the scratch space. This field is only valid when Scratch Space is enabled. Format = GeneralStateOffset[31:10]
	9	Reserved: MBZ
	8:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by the thread upon thread completion.

DWord	Bits	Description
R0.4	31:5	**Binding Table Pointer.** Specifies the 32-byte aligned pointer to the Binding Table. It is specified as an offset from the **Surface State Base Address**. Format = SurfaceStateOffset[31:5]
	4:0	Reserved: MBZ
R0.3	31:5	**Sampler State Pointer.** Specifies the 32-byte aligned pointer to the sampler state table. Format = GeneralStateOffset[31:5]
	4	Reserved: MBZ
	3:0	**Per Thread Scratch Space.** Specifies the amount of scratch space, in 1K-byte quantities, allowed to be used by this thread. The value specifies the power to which two is raised to determine the amount of scratch space. Format = U4 Range = [0,11] indicating [1K bytes, 2M bytes] in powers of two
R0.2		
R0.1	31:28	Reserved: MBZ
	27	Reserved
	26:25	Reserved: MBZ
	24:16	**Scoreboard Y.** This field provides the Y term of the scoreboard value of the current thread. Format = U9
	15:11	Reserved: MBZ
	10:9	Reserved: MBZ
	8:0	**Scoreboard X.** This field provides the X term of the scoreboard value of the current thread. Format = U9
R0.0	31:30	Reserved: MBZ
	29:28	Reserved: MBZ
	27:24	**SLM ID.** This field indicates the index to the SLM starting offset associated with this kernel. Format: U4
	23:16	**Scoreboard Mask.** Each bit indicates the corresponding dependency that the scoreboard is dependent on. This field is AND'd with the corresponding Scoreboard Mask field in the MEDIA_VFE_STATE. **Bit n (for n = 0..7)**: Scoreboard n is dependent, where bit 23 maps to n = 0. Format = TRUE/FALSE
	15:0	**URB Handle.** This is the URB handle where indicating the URB space for use by the root thread and its children.

Thread Spawn Message

The thread spawn message is issued to the TS unit by a thread running on an EU. This message contains only one 8-DWord register. The thread spawn message may be used to:

- Spawn a child thread.
- Spawn a root thread (start dispatching a synchronized root thread).
- Dereference an URB handle.
- Indicate a thread termination, dereference other TS managed resource and may or may not dereference URB handle.
- Release a PRT_Fence.

To end a root thread, the end of thread message must be targeted at the thread spawner. In this case, the root thread sends a message with a "dereference resource" in the Opcode field. The thread spawner does *not* snoop the messages sideband to determine when a root thread has ended. Thread Spawner does not track when a child thread terminates, to be consistent a child thread should also terminate with a "dereference resource" message to the Thread Spawner. Software must set the Requester Type (root or child thread) field correctly.

TS dispatches one synchronized root thread upon receiving a 'spawn root thread' message (from a synchronization thread). The synchronizing thread must send the number of 'spawn root thread' message exactly the same as the subsequent 'synchronized root thread'. No more, no less. Otherwise, hardware behavior is undefined.

URB Handle Offset field in this message (in M0.4) has 10 bits, allowing addressing of a large URB space. However, when a parent thread writes into the URB, it subjects to the maximum URB offset limitation of the URB write message, which is only 6 bits (see Unified Return Buffer Chapter for details). In this case, the parent thread may have to modify the URB Return Handle 0 field of the URB write message to subdivide the large URB space that the thread manages.

Only a persistent root thread can use this message to dispatch a root thread if preemption exceptions are possible. The root thread requested by this message is not guaranteed to dispatch, and the persistent root thread must handle the case where it does not dispatch. When a context switch interrupt is recognized by the persistent root thread, all other root threads that had been dispatched have completed and no more will be dispatched.

Child threads requested by this message are guaranteed to dispatch in all cases, so long as the persistent root thread does not also dispatch synchronized root threads. A child thread does not dispatch if it is behind a synchronized root thread that is not dispatched due to a preemption exception.

In addition to monitor 'end of thread message' targeted to Thread Spawner, Thread Spawner also monitors the message targeting to Message Gateway for EOT signal. Therefore, a child thread, who doesn't hold any hardware resource (URB handle or scratch memory) that Thread Spawner manages, can terminate with a Gateway message with EOT on. The reason of this new TS feature is to avoid a possible risk condition as described below.

In a system running child threads, a parent thread is monitoring the status of the child threads by communications through Message Gateway. When a child thread is about to terminate, it sends a message to the parent through Message Gateway and then sends a second message of EOT (end of thread) to TS.

There is a latency between sending a message to parent thread and the EOT to TS due to message bus arbitration. The parent thread may acknowledge the GW message and issue a new child dispatch before the EOT was processed; basically threads are issued faster than retired.

Because the messages for new child dispatch and EOT go to the same queue in TS, if the queue gets full, EOTs will get blocked. In the case when all the EUs/Threads are full, this will create a system deadlock: no EOTs can be acknowledged by TS (to free up EU resource) and no child threads can be dispatched (to free up TS queue to receive EOT message).

Message Descriptor

The following table shows the lower 20 bits of the message descriptor within the SEND instruction for a thread spawn message.

Thread Spawn Message Descriptor

Message Payload

DWord	Bits	Description
M0.5	31:8	Ignored.
	7:0	**FFTID.** This ID is assigned by TS and is a unique identifier for the thread in comparison to other concurrent root threads. It is used to free up resources used by a root thread upon thread completion. This field is valid only if the **Opcode** is “dereference resource”, and is ignored by hardware otherwise.
M0.4	31:16	Ignored.
	15:10	**Dispatch URB Length.** Indicates the number of 8-DWord URB entries contained in the Dispatch URB Handle that will be dispatched. When spawning a child thread, the URB handle contains most of the child thread's payload including the R0 header. When spawning a root thread, the URB handle contains the message passed from the requesting thread to the spawned “peer” root thread. The number of GRF registers that are initialized at the start of the spawned child thread is the sum of this field and the number of URB constants if present. The number of GRF registers that are initialized at the start of a spawned root thread is the sum of this field, the number of URB constants if present, and the URB handle received from VFE. This field is ignored if the Opcode is “dereference resource”. A Length of 0 can be used while spawning child threads to indicate that there is no payload beyond the required R0 header. A Length of 0 while spawning a root thread indicates that there is no payload at all from the parent thread. A spawned root has R0 supplied by the Media_Object command indirect/inline data.

DWord	Bits	Description
		Format = U6 Range = [0,63] for child threads.
	9:0	**URB Handle Offset.** Specifies the 8-DWord URB entry offset into the URB handle that determines where the associated dispatch payload will be retrieved from when the spawned child or root thread is dispatched. This field is ignored if the **Opcode** is "dereference resource". Format = U10 Range = [0,1023]
M0.3	31:0	Ignored.
M0.2	31:28	Ignored.
	27:24	**BarrierID.** This field indicates which one of the 16 Barriers this kernel is associated with. Format: U4
	23:16	Ignored.
	15:10	Ignored.
	3:0	**Scoreboard Color** (only with MEDIA_OBJECT_EX). This field specifies which dependency color the current thread belongs to. It affects the dependency scoreboard control. Format = U4
M0.1	31:0	Ignored.
M0.0	31:28	Ignored.
	27:24	**Shared Local Memory Index.** Indicates the starting index for the shared local memory for the thread group. Each index points to the start of a 4K memory block, 16 possibilities cover the entire 64K shared memory per half-slice. Format = U4
	23:16	Reserved: MBZ
	15:0	**Dispatch URB Handle.** If **Opcode** (and **Requester Type**) is "spawn a child thread": Specifies the URB handle for the child thread. If **Opcode** (and **Requester Type**) is "spawn a root thread": Specifies the URB handle containing message (e.g. requester's gateway information) from the requesting thread to the spawned root thread. If **Opcode** is "dereference resource": This field is required on end of thread messages if the **Children Present** bit is set, as the handle must be dereferenced, otherwise this field is ignored.

L3 Cache and URB

This section describes the GFX L3 Cache, which is a large storage that backs up various L2/L1 caches on many clients. It provides a simple way based partitioning option for each of a cluster of clients to get a dedicated chunk of the cache. It also acts as a GFX URB and can be configured as highly banked memory for EUs/ROWs.

L3 Cache and URB

This section discusses GFX L3 cache for CHV, BSW. The included topics are:

- Overview
- Atomics
- L3 Coherency
- L3 Allocation & Programming
- Shared Local Memory (SLM)
- Dynamic Parity Feature for GFX L3 Cache

Vol 1i L3$/URB

Overview (8th Generation)

GFX L3 cache is introduced for Gen7 GFX core as a large storage which backs up various L2/L1 caches on many clients. It provides a simple way based partitioning option for each or a cluster of clients to get a dedicated chunk of the cache. It also acts as a GFX URB and can be configured as highly banked memory (shared local memory) for EUs/ROWs.

For CHV, BSW which is a generation 8 derivative, the bank count has been reduced to 2 (i.e. 2x192KB)

- Formed as 2 individual banks each with 192KB in size
- Each logical bank consists of
  - Data Array
  - Tag Array
  - LRU Array (implements a Pseudo Least Recently Used algorithm)
  - State Array
  - SuperQ Buffer
  - Atomic Processing Units
- The rest of the support logic around L3 are
  - SuperQ (main scheduler)
  - Ingress/Egress queues to L3/SQ (L3 arbiter)
  - CAM structures to maintain coherency.
  - Crossbars for data routing
- Use of 2x/1x clocking
- A portion of L3 can be allocated as highly banked memory and/or unified buffer (URB)

Gen8 L3 controller added an optimization where back 2 back accesses to the same address are performed full bandwidth rather than partial serialization that was on previous generations. This optimization eliminates the need for software work around of distributing or moving of high accesses buffers within a short temporal period.

L3$ Cache Configuration

- 2x192KB cache, 96 logical ways
  - Upto 64 ways tagged for L3$, remaining is treated as memory.
  - Shared local memory capable (up to 256KB per slice, i.e. GT2)
- 64B Cacheline with a portion capable of highly banked memory (with 16x4B)
- Interface 2x64B to SQDB for the fill/write path, 2x64B Read/Evict path to SQDB. Additional 2x64B read and 2x64B write capability for SLM
- Data Array built via 6T cells
  - Data protection via parity/ECC
- TAG/LRU/STATE (using gen-ram via RLS flows)
  - 39-bit addressing support in TAG
  - 6 bit state
  - Intel pseudo-LRU implementation for selecting the line to be replaced
- Repetition rates for each operation
  - All operations – 1 every 2x clock
    - With b2b restriction for same type of accesses (i.e. read to read or write to write)

Blocks(s) Overview

L3 is formed via 2 logical banks that are very similar to each other in formation. The major blocks in each logical banks

- L3 Cache Arrays & Controller
- Super Q and related data buffer
- Ingress queues and related CAMs with arbitration
- Atomics Block/SLM pipeline & crossbar for data routing

Rest of the document will go through the details of these blocks and provide details of their logical operation. In addition there will be specific sections that will go through the requirements for coherency.

Vision is to build a compute scalable cache where with each additional compute both the size and bandwidth is scaled while maintaining the functional single cache concept. Each added bank becomes an additional cache rather than an independent content. The concept is to be able to keep a single copy of a line and service all requesters via distributing their accesses over many physical caches.

Size of L3 for 8th Generation

Non-SLM Mode		
	GT1 (384KB)	GT0.5 (192KB)
URB	128KB-192KB	64KB-96KB
Shared local Mem	0KB	0KB
L3$	192KB-256KB	96KB-128KB

SLM Mode		
	GT1	GT1
URB	64KB	32KB
Shared local Mem	128KB	64KB
L3$	192KB	96KB

ECC Protection

8th generation L3 has added ECC (SECDED) protection on a 32B level to provide better data protection. The algorithm is an industry standard. Double bit errors are reported to s/w.

L3$ Theory of Operation

L3/URB operation is required for various clients to access L3 as their back-up cache or memory space. The following clients are listed as L3/URB clients.

L3 Clients:

- Data Cluster (i.e. spill/fills, load/stores, global memory accesses, ...) (read/write)
- Sampler (L2$ - MT) (read)
- IME (motion estimation) (read)
- I$ (Instruction Cache) (read)
- State Arbiter (L3 is state cache replacement) (read)
- Constant cache (read)
- Tesselator (always un-cacheable for SW tessellation)

URB Clients:

- TD-L (read client) – Local Thread Dispatcher
- SBE (read client) – SF Backend
- SOL (read client) – Stream Out
- CL (read client) – Clipper
- GS (read client) – Geometry Shader
- TE (read Client) – Tessalator

- DC (read/write client) – Data Cluster
- VF/(VFE/CS) (write client) – VF acts on behalf of all

L3 vs. URB accesses are separated with a simple field on the request field; for most clients this is only one direction with the exception of the data port where both could be addressed L3 or URB. The destination field is part of the request and could be set to redirect the request to:

1. L3 cache
2. URB
3. State Arbiter
4. SLM (highly banked memory)

L3 access/cacheability is determined via request field as well; such parameter will be part of the surface state or base address programming of L3 clients and will be communicated to L3 Cluster along with the request packet.

Atomics

An atomic operation may involve both reading from and then writing to a memory location. Atomic operations apply only to either u# (Unordered Access Views) or g# (Thread Group Shared Memory). It is guaranteed that when a thread issues an atomic operation on a memory address, no write to the same address from outside the current atomic operation by any thread can occur between the atomic read and write.

If multiple atomic operations from different threads target the same address, the operations are serialized in an undefined order. This serialization occur due to L3 serialization rules to the same address.

Atomic operations do not imply a memory or thread fence. If the program author/compiler does not make appropriate use of fences, it is not guaranteed that all threads see the result of any given memory operation at the same time, or in any particular order with respect to updates to other memory addresses. However atomic operations are always stated on a global level (except on shared local memory), when atomic is operation is complete final result is always visible to all thread groups.

On Gen7, Atomicity is implemented at 32-bit granularity. If a load or store operation spans more than 32-bits, the individual 32-bit operations are atomic, but not the whole.

Gen7.5 extended the atomic support with the addition of “Atomic_CMP/WR8B” where 64-bit atomic operations are introduced. An additional change was to move the global atomics to GPU interface to keep the operation coherent between GPU and IA cores. In addition full 64B atomic access option was provided to both SLM and Global atomics to increase the throughput. Gen7.5 (GT3 option) can support up to 128 atomic ops per clock on SLM and 16 atomic ops per clock on Global memory (8B and compare exchange atomics are subject to less due to amount of data needs to be carried)

Gen8 atomics has following features introduced:

- Double size operands where 8B atomic operations are introduced for 64-bit data types. There is also an addition 16B "atomic_cmp/wr16B" to the table. All these new operands are applicable to global memory only
- Floats for 4B/8B accesses, only floating point adder is used.
- Move global atomic ops to L3 (keep the GTI support for non-coherent L3 mode) and share same atomic OPs as SLM.

Programming Note	
Context:	Atomics
Atomic operations on Thread Group Shared Memory are atomic with respect to other atomic operations, as well as operations that only perform reads ("load"s). However atomic operations on Thread Group Shared Memory are NOT atomic with respect to operations that perform only writes ("store"s) to memory. Mixing of atomics and stores on the same Thread Group Shared Memory address without thread synchronization and memory fencing between them produces undefined results at the address involved. This restriction arises because some implementations of loads and stores do not honor the locking semantics for implementing atomics. It turns out this has no impact on loads, since they are guaranteed to retrieve a value either before or after an atomic (they will not retrieve partially updated values, given they are all defined at 32-bit quanta). However store operations could find their way into the middle of an atomic operation and thus have their effect possibly lost.	

In L3 or SLM, the atomic operation leads to a read-modify-write operation on the destination location with the option of returning value back to requester. The table below is defined as a list of atomic operations needed:

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
Atomic_AND	0000_0001	Single component 32-bit bitwise AND of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" AND "src0"	global/SLM	old_dst
Atomic_OR	0000_0010	Single component 32-bit bitwise OR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" OR "src0"	global/SLM	old_dst
Atomic_XOR	0000_0011	Single component 32-bit bitwise XOR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" XOR "src0"	global/SLM	old_dst
Atomic_MOVE	0000_0100	Replacement of the dst with src0.	"src0"	global/SLM	old_dst
Atomic_INC	0000_0101	Single component 32-bit integer increment of dst back into dst	"old_dst + 1"	global/SLM	old_dst

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
Atomic_DEC	0000_0110	Single component 32-bit integer decrement of dst back into dst	"old_dst - 1"	global/SLM	old_dst
Atomic_ADD	0000_0111	Single component 32-bit integer add of operand src0 into dst at 32-bit per component address performed atomically. Insensitive to sign	"old_dst + src0"	global/SLM	old_dst
Atomic_SUB	0000_1000	Single component 32-bit integer subtraction of operand src0 into dst at 32-bit per component address performed atomically. Insensitive to sign	"old_dst - src0"	global/SLM	old_dst
Atomic_RSUB	0000_1001	Single component 32-bit integer subtraction of operand dst from src0 into dst at 32-bit per component address performed atomically. Insensitive to sign	"src0 - old_dst"	global/SLM	old_dst
Atomic_IMAX	0000_1010	Single component 32-bit signed MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMAX (old_dst, src0)	global/SLM	old_dst
Atomic_IMIN	0000_1011	Single component 32-bit signed MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMIN (old_dst, src0)	global/SLM	old_dst
Atomic_UMAX	0000_1100	Single component 32-bit unsigned MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMAX (old_dst, src0)	global/SLM	old_dst
Atomic_UMIN	0000_1101	Single component 32-bit unsigned MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMIN (old_dst, src0)	global/SLM	old_dst
Atomic_ CMP/WR	0000_1110	Single component 32-bit value compare of operand src0 with dst at 32-bit per component address dstAddress	(src0 == old_dst)?	global/SLM	old_dst
		If the compared values are identical, the single-component 32-bit value in src1 is written to destination memory, else the destination is not changed	src1:		
		The entire compare+write operation is performed atomically	old_dst		
Atomic_	0000_1111	Single component 32-bit integer	"old_dst - 1"	global/SLM	new_dst

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
PREDEC		decrement of dst back into dst			
Atomic_AND8B	0010_0001	Single component 64-bit bitwise AND of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" AND "src08B"	global	old_dst8B
Atomic_OR8B	0010_0010	Single component 64-bit bitwise OR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" OR "src08B"	global	old_dst8B
Atomic_XOR8B	0010_0011	Single component 64-bit bitwise XOR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" XOR "src08B"	global	old_dst8B
Atomic_ MOVE8B	0010_0100	Replacement of the dst with src0.	"src08B"	global	old_dst8B
Atomic_INC8B	0010_0101	Single component 64-bit integer increment of dst back into dst	"old_dst8B + 1"	global	old_dst8B
Atomic_DEC8B	0010_0110	Single component 64-bit integer decrement of dst back into dst	"old_dst8B - 1"	global	old_dst8B
Atomic_ADD8B	0010_0111	Single component 64-bit integer add of operand src0 into dst at 64-bit per component address performed atomically. Insensitive to sign	"old_dst8B + src08B"	global	old_dst8B
Atomic_SUB8B	0010_1000	Single component 64-bit integer subtraction of operand src0 into dst at 64-bit per component address performed atomically. Insensitive to sign	"old_dst8B - src08B"	global	old_dst8B
Atomic_ RSUB8B	0010_1001	Single component 64-bit integer subtraction of operand dst from src0 into dst at 64-bit per component address performed atomically. Insensitive to sign	"src08B - old_dst8B"	global	old_dst8B
Atomic_IMAX8B	0010_1010	Single component 64-bit signed MAX of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMAX (old_dst8B, src08B)	global	old_dst8B
Atomic_IMIN8B	0010_1011	Single component 64-bit signed MIN of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMIN (old_dst8B, src08B)	global	old_dst8B

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (optional)
Atomic_ UMAX8B	0010_1100	Single component 64-bit unsigned MAX of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	UMAX (old_dst8B, src08B)	global	old_dst8B
Atomic_ UMIN8B	0010_1101	Single component 64-bit unsigned MIN of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	UMIN (old_dst8B, src08B)	global	old_dst8B
Atomic_ CMP/WR8B	0010_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress	(src08B == old_dst8B)?	global	old_dst8B
		If the compared values are identical, the single-component 64-bit value in src1 is written to destination memory, else the destination is not changed	src18B:		
		The entire compare+write operation is performed atomically	old_dst8B		
Atomic_ PREDEC8B	0010_1111	Single component 64-bit integer decrement of dst back into dst	“old_dst8B - 1”	global	new_dst8B
Atomic_ CMP/WR16B	0100_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress	(src0_16B == old_dst16B)?	global	old_dst16B
		If the compared values are identical, the single-component 64-bit value in src1 is written to destination memory, else the destination is not changed	src1_16B:		
		The entire compare+write operation is performed atomically	old_dst_16B		

L3 Coherency

Coherency is one of the crucial topics within L3; there are multiple levels of coherency that are checked and ensured via L3. The list of domains and flows depends on the usage models, however the basic premise is always the same.

The coherency levels:

1. Thread Level Coherency within a Thread Group
2. Thread Group Coherency between multiple domains

Besides these special domains, basic producer/consumer models are followed, which are listed as:

1. Fixed function is producing.
2. Data Port is producing.

All these high level coherency models are investigated and addressed.

Note: GPU/IA level coherency is not supported by CHV, BSW Hardware. This level of coherency must be enforced via Software, same as Gen 7.5.

Thread level Coherency

A given thread group is contained within a quarter slice in Gen8, where its writes and reads target the L3 for global memory and SLM for shared local memory. Given the shared local memory view is same for all quarter slice accesses there is no question about the coherency or data sharing within the thread group. Local syncs are executed up to HDC boundary and not exposed to L3.

Thread Group Coherency

Thread groups can be distributed to multiple quarter slices that are physically far from each other. The coherency between thread groups can only be maintained for their global memory accesses. There are two implications of this coherency depending on the mode we are operating at:

1. Non IA-Coherent L3 mode: This is the same methodology that was introduced on Gen7.5 with the addition of GT4 support. The cross thread group coherency is maintained via Sync Global which is processed by L3 as well as introducing WT mode to be able to update global memory with latest data.
2. IA-Coherent L3 mode: For the new mode of operation, there is no need to have WT behavior. The data in L3 is already visible to all consumers (i.e. other thread groups of GPU or IA cores). Sync'global has no affect given the data in L3 is already globally visible to all consumers.

GPU/IA level Coherency

In non-coherent L3 mode (i.e. Gen7.5 behavior), data sharing between GPU and IA happens via a SW controlled flow which requires the internal GPU caches to be flushed in order to make the data visible; similarly same caches need to be invalidated when IA produces data.

Coherency Usage Models

This section is to give some examples of usage models and high level handling within L3. They are specific to L3 flows and not meant to represent over coherency usage models on the system level.

Fixed Func. Producing (URB)

Fixed functions clients and slice clients consuming is a very common usage model for URB based data sharing. In this mode, FF sends writes to URB and shifts to their pipeline, rest of the pipeline clients including slice clients will read their content from URB.

Process is achieved via URB writes completion from the GO point which would be the *slice node* for gen8. Once slice node consumes the write and schedules for BANK the completion is returned back to producer which enables the consumer. The time consumers start, the data is already at the point of GO where BANK superQ will keep the ordering.

Note that a node should not be scheduling writes from ingress queues if there are no credits and no progress can be made. The completion from a node is only given once the write from the producer leaves the node ingress queue.

Fixed Func. Producing (Push Constants)

Push constants are also similarly processed, in fact the target is the same: URB. However URB accesses are processed with their URB offsets via TAG, their original address is based on virtual memory pointing to buffer location in memory. The delivery mechanism makes the CAMs not possible hence the completions are managed once access is deemed towards SuperQ of the corresponding L3 Bank.

There are additional constraints independent L3 handling of push constants which are defined as part of the Global Arbitration fabric.

EUs Producing via HDC

Data port is the producer for Global and Shared Local memory types. The local memory is specific to a data port, hence L3 does not have to do anything special to maintain coherency but to keep accesses to SLM in-order.

For Global memory coherency is maintained via combination of many mechanisms.

1. Completion tracking: Each HDC tracks their writes towards L3 and wait for them to reach to GO. GO message is given by L3

a. For Non-coherent/virtual addressed Write: Once the ingress queue retires the write in the corresponding node to targeted bank.

b. For Coherent/physical addressed Write: Once the ownership is obtained for the write (i.e. read-for-ownership or invalid-to-modified) which means write is GO with respect to IA cores.

GO information is used within data port to make sure handle releases are gated until the producers updates are globally visible.

2. Thread Level Flush: EU threads have the capability to push globally tagged data from L3 to next level caches.

Invalidation and Flushes

Cache invalidation and flush flows need to analyzed and defined as the u-arch changes for gen8. Fundamentally there are two different sources to initiate these sequences:

- Command Streamer initiated
- EU/thread initiated

Both cases need to be analyzed for two modes of operation

- Non-IA coherent L3

In addition there are side flows that back up the various flows and they do require invalidation/flush sequences to be executed for their purpose. Their behavior has no impact between two modes of L3 operation.

- Global Invalidation
- Power Management Invalidation

Node Architecture Impact

The idea behind the node architecture is to make each L3 node to be independent and easily scalable; this is done via standardizing the flows and using common interfaces to exchange information between blocks. Invalidations and Flushes are meant to be tuned to similarly to fit into the concept.

Each node will have to be notified on all invalidations as soon as possible (signals) and their behavior is to make that particular stream un-cacheable even if it did not start the invalidation process already. The process is much simpler when it comes to operations which do not require a feedback back to originator. The command streamer top of the pipe invalidations are under this category. Each node can process these invalidations independently but act on it for turning the particular stream to look uncacheable right away.

For the invalidation and flush processes where feedback is required, the event needs to be synchronized between nodes.

Each node receiving the invalidation or flush will push their ingress FIFOs to make sure their content is also covered as part of the flush event.

Command Streamer Flows

Command streamer can do top of the pipe invalidations which are direct connections from command streamer unit to L3 in respective slices, as well as pipeline flush which runs through the data port to respective slice L3 node. Both flows are handled separately.

There is no definition of a particular context using a coherent vs non-coherent L3. Hence L3 has no register bit that says it is running in coherent vs non-coherent mode. In the operation time, both modes

will co-exist simultaneously where some lines in the cache are following a coherent protocol (i.e. physical address) and remaining lines are following non-coherent protocol (i.e. virtual address).

Non-IA Coherent Flows

This is the traditional flow where the content of L3 needs to be invalidated or flushed similar to gen7.5 flows.

Top of the Pipe Invalidations

The nodes process invalidations independently. Each top and bottom slice (slice0 and slice1 OR slice2 and slice3) will get their top of the pipe invalidations from their command streamer. They will process them independently (via communicating to L3 blocks) and getting a feedback for completion. They will also communicate the invalidation to across cores (via horizontal communication).

Nodes are responsible to serialize the invalidation and use double buffering for each event.

End of the Pipe Flush - Pipeline Flush

GT4/GT3/GT2/GT1 pipeline invalidation is much simpler where the pipeline flush needs to be coordinated between two slice or not need to be coordinated at all.

End of the Pipe Flush - GT3/GT4

GT3 has only two/three slices stacked, the flush event needs to be coordinated between two slices. Once both nodes receive the flush, they will perform what is needed and report back their status.

End of the Pipe Flush - GT2/GT1

GT2 and GT1 has a single slice hence no cross communication dependency is needed.

IA-Coherent Flows

In principle IA-coherent flow is same from node perspective, the only difference is within the banks given the data that we are trying to flush or invalidate is already coherent with IA. Such case eliminates the need to push any bank content explicitly out to LLC.

All that is needed from the bank is to make sure LSQC content is posted into the bank where it is IA visible for a bank to return flush/invalidation completion status to the node.

EU/Thread Flows

As part of Gen7.5, we added the capability for EUs to be able to perform flush/invalidation events that are not coordinated between other EUs. HDC will relate the request to corresponding L3 and require the entire content of this buffer to be invalidated or flushed to relative coherency domain.

Similar to pipeline flush and top of the pipe invalidations, each node receiving this event will have to communicate with other nodes and make the corresponding traffic uncacheable before returning the

response back to corresponding HDC, allowing progress. The actual invalidation/flush will be deferred and performed once all nodes agree on what needs to be done.

The IA-coherent vs non-IA coherent treatment is same as command streamer flows and still applicable for EU/Thread Flows for invalidation/flush.

Global Invalidation

As part of a mitigation plan to plug holes, a global invalidation flow is introduced where each L3 will get the request from their SARB (or config agent) and again coordinate between the nodes. Once invalidation/flush is performed, completion will be returned back to config agent to clear the flags allowing SW to observe the end of the global invalidation.

Global invalidation will invalidate/flush the entire content of L3 regardless of the coherency mode.

Power Management Invalidation

Corresponding PM will message to L3 config agent to request a flush when needed. Main usage mode is prior to entering RC6. The rest of the treatment in the nodes is the same. Once flush/invalidate event is complete, message(s) will be sent back to PM with the completion status.

L3 Allocation and Programming

L3 Cache allocation is done on a per way basis which should be consistent across all 4 banks (2 banks for GT1). The way allocation between URB and any of the L3 clients can only be changed post pipeline flush where L3 contains no data. This is required for stream based flushes to be dependent on the way allocation of these corresponding streams. S/W should not be removing ways under a particular stream and expect a later pipelined stream flush to target all the corresponding locations. The stream based flush will be performed on the existing way allocation of that stream, there is no history of previous way allocation tracked in the hardware.

L3 Cache has been divided into following client pools:

- Shared Local Memory: When enabled its size is always fixed to 128KB (64KB for GT1)
- URB: Local memory space, provides a flexible allocation on per 8KB granularity
- DC: Data Cluster Data type
- Inst/State: Both instructions and state allocation is combined
- Constants: Pull constants for EUs
- Textures: texture allocation to back-up L2$

In addition to these sub-groups, a collection of groups are generated to bundle multiple clients under the same allocation set:

- All L3 Clients: DC, Inst/State, Constants & Textures
- Read-Only Clients: Inst/State, Constants & Textures

Each of the L3 way allocations are managed via pLRU, hence best performance can be attained via assigning a power-of-2 number of ways. This is to ensure pLRU to distribute the ways w/o hot spotting within that client's group. Even though design provides a flexible (per way basis) programming model for way allocation for each client following table is given for validation and s/w programming models. The programming options in the following table represents most likely cases for different operation modes.

For GT1, hardware will retain 2 of the L3 banks hence all following allocations will be reduced half the size.

The URB size programming given in this section shows L3 capability but does not necessarily dictate the s/w programming requirements. The programming should be based on what s/w chooses in 3d command for 3DState_URB_* selection. Once the URB size is determined, the remaining area can be given to L3$.

Non-SLM Mode Allocation

Normal L3/URB mode (non-SLM mode), uses all 4 banks of L3 equally to distribute cycles. The following allocation is a suggested programming model. Note all numbers below are given in KBytes.

The following settings are given for GT1.

GT0.5 has the half of the allocation listed below.

Normal Banked - No SLM									
	SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
0	0	192	192	0	0	0	0	0	384
1	0	192	0	64	128	0	0	0	384
2	0	128	0	64	192	0	0	0	384
3	0	128	0	0	256	0	0	0	384
4	0	128	256	0	0	0	0	0	384

SLM Mode Allocation

With the existence of Shared Local Memory, a 64KB chunk from each of the 2 L3 banks will be reserved for SLM usage. The remaining cache space is divided between the remaining clients.

Shared Local Memory Mode - highly banked Memory									
	SLM	URB	Rest	DC	RO(I/S/C/T)	I/S	C	T	Sum
1	128	64	192	0	0	0	0	0	384
2	128	64	0	64	128	0	0	0	384
3	128	64	0	128	64	0	0	0	384

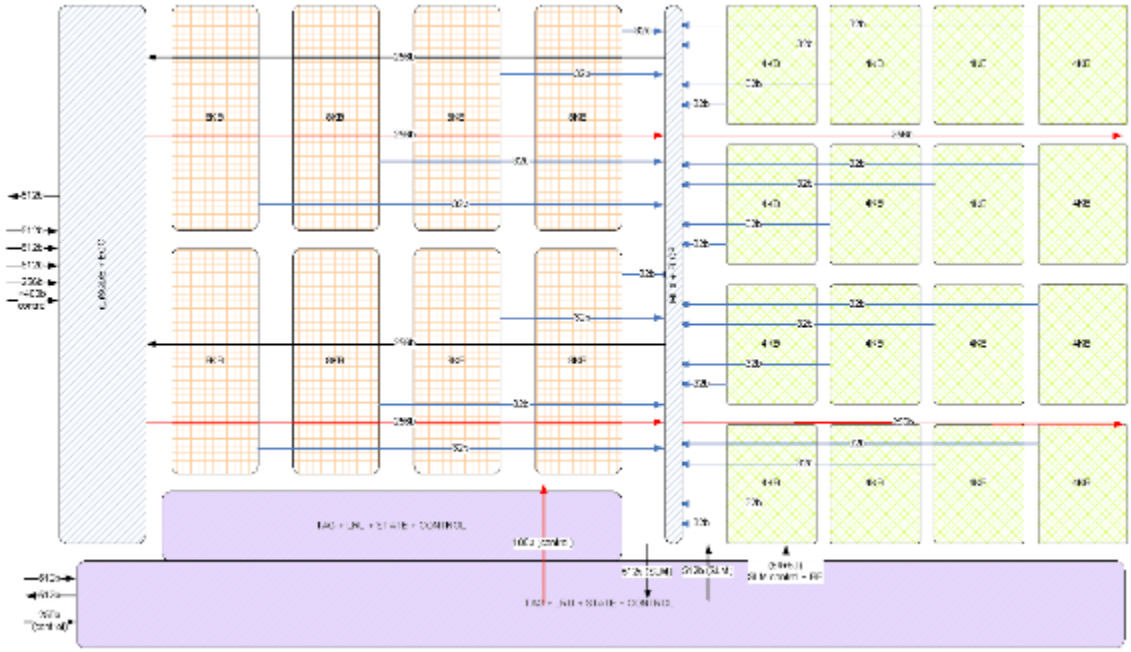
Unlike previous generation logic is simplified via combining RO clients under one pool, given most clients have already their L1 and L2 caches. Bundling them greatly reduces the timing paths and complexity in trying to determine which client belongs to which way.

Similar to previous generations before changing the configuration of L3, the entire contents needs to be flushed and invalidated. This is to prevent any false hits post configuration update. The invalidation can be achieved either through command streamer's pipeline flush + top of the pipe invalidations or the global invalidation option within L3. Both options will work for non-coherent option of L3.

For coherent L3 the only option is to follow pipeline invalidation via an L3 global invalidation. That would push all coherent content out of L3 as well allowing a programming update in allocation.

Shared Local Memory (SLM)

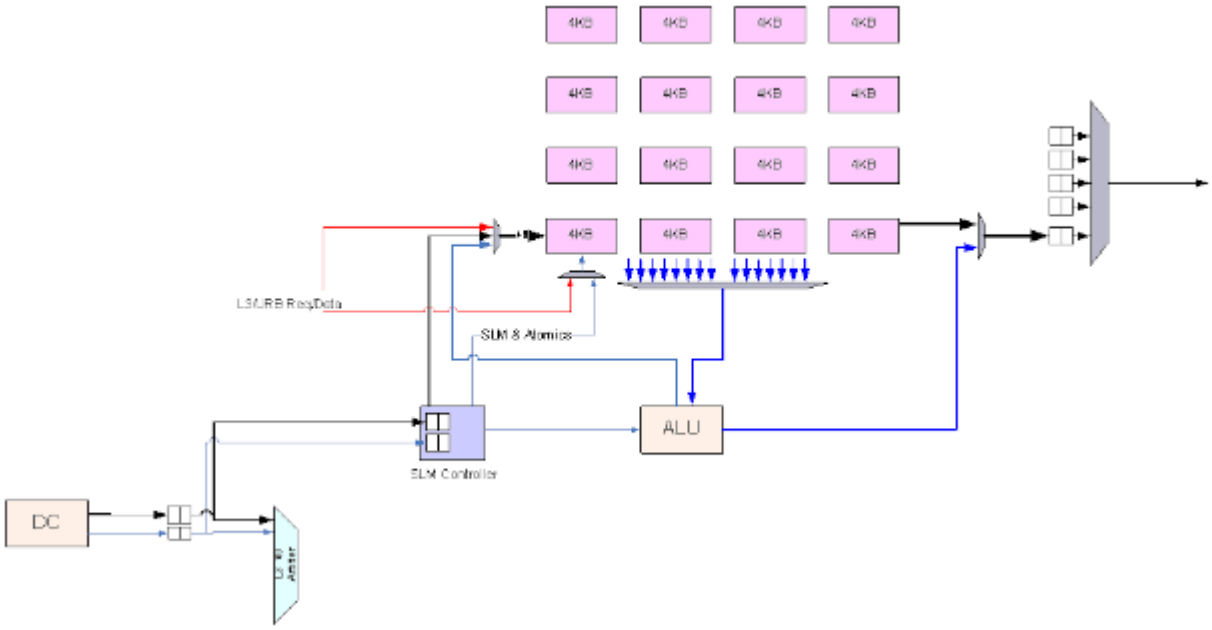
Shared local memory (aka highly-banked memory) is a portion of L3 which will be dedicated to EUs as a local memory when enabled. The accesses are only possible through data cluster with the destination flag set as SLM. In order to support a highly banked design, 2 of the L3 banks are structured to have 16x4KB portion which could be accessed independently per clock. This part of the L3 can support 16 DW size accesses (per SLM) in a given clock cycle.



These 16 banks can either be used as L3/URB or used as shared local memory with parallel accesses to all banks. The choice of enabling SLM mode is done through MMIO programming.

Bits	Access	Default Value	Description
0	RW/C	0	Enable Shared Local Memory: When set, it enables the use of 2 banks of L3 as shared local memory which allows 64KB of L3 to be banked as 16x4KB and allows independent accesses to all banks within the same clock cycle. Note: This mode can only be enabled once L3 content is completely flushed.

In gen7, SLM is structured as 64KB per half slice (6 EUs) and each 64KB allow up to 16 accesses to a particular 4KB bank. Each 4KB bank is addressed separately where their requests are placed on a 160bit address bus where each 10-bit correspond to a bank sequentially. Each bank gets addressed with 10bits and provides 4B access with a total of 4KB per bank. When SLM mode is not enabled, SLM banks are considered a part of L3 and used for cache or URB.



SLM requests are forked around the L3 arbiter, post ingress FIFOs for DC. L3 arbiter delivers request/data to SLM controller upon the availability of credits. Request will be crossed to 2x clock domain routed to corresponding banks. Individual bank controls are managed via SLM controller which are muxed with L3/URB accesses. Note that SLM accesses do carry byte enables and needs to be honored towards the banks. If the request has atomic requirements, SLM controller will provide the data to ALU along with the atomic type. Output data is again managed with SLM controller towards the output cross bars.

SLM should not be accessed through the 3D pipe.

Save and Restore Requirements

At the pre-emption time for GPGPU the contents of shared local memory will be saved to memory. The flow requires some special handling both in SuperQ and L3 pipeline and will be managed via a small DMA engine which will need to interface to L3 arbiter.

The SAVE time will be reported via messages to state arbiter which later relates that message to DMA controllers on either side for GT2 (only one for GT1), restore will be processed similar to save and will be communicated back as completed.

Note that GFX virtual buffer given for save and restore has to be 128KB aligned (64KB for GT1).

As part of the process of preemption, the GPGPU controller will ensure there is no more messages from EUs to shared local memory. Once guaranteed, there will be a message to State Arbiter which in turn converted to a message to SLM DMA controller block. From this point on SLM DMA controller will interface with L3 arbiter like any other GFX client with the credit protocol that are defined.

Once all requests are retired to L3 arbiter, DMA controller will send a completion signal to SARB. State arbiter will wait for both DMA controllers to send their completion signals before sending the response back to GPGPU controller (note for GT1 – I response is good enough)

A new simple DMA controller will be designed to generate the accesses that are needed, this block will be used for both save and restore transaction.

For Save, once the signal is received from State Arbiter to kick-off the save along with the address within the gfx virtual space, DMA controller will start generating following special cycles towards L3 arbiter. Similarly for restore another special request will be sent.

- Save SLM Request
- Restore SLM Request

The request to L3 controller will be 64B size and carry a gfx address similar to any other request that L3 arbiter supports. DMA controller will walk all 64KB address space (1024 entries) via an increment and using the provided address as the base address for request to L3 arbiter.

The interface between the SLM DMA controller and L3 arbiter will be no different than any other client which is a credit based protocol.

Once the Save or Restore cycles are all posted to L3 arbiter, DMA controller will signal back to State Arbiter to send the ACK message back to requester.

In L3 arbiter, the save and restore requesters from DMA controller is treated as any other request interface. There is no data involvement so only request queues are needed. A 4 deep structure (ingress FIFO) is provided giving 4 credits to SLM DMA controller to consume.

L3 arbiter will mux in the new ingress FIFO output with DC read paths and arbitrate Fixed Priority (SLM DMA controller > DC reads).

The only remaining requirement for L3 arbiter to ensure the flush of this ingress FIFO upon a pipelined Flush received from DC.

In processing save and restore cycles existing flows will be overloaded in SuperQ, both the save and restore request is treated slightly different in SQ FSM.

SLM Context Save in SuperQ

Context Save request is a special request that SQ will generate towards L3 pipeline. They will look like URB reads however will use the destination information for SLM. The GFX address is passed along to L3 to locate the line being asked.

SuperQ will eventually get a data response from L3 stating the readiness of line in SQDB. This will be followed with a write request to Pixel arbiter (through crossbar) similar to eviction writes.

6SLM Context Restore in SuperQ

Context Restore for SLM will be also a special cycle and will look like prefetch-to-URB. Where SQ will read the corresponding GFX address from memory and bring the data into SQDB. Than it will follow with a write request to SLM – this is similar to write to URB however it will have the SLM as destination.

Similar to SuperQ, L3 pipeline will have some slight differences for SLM requests coming from the SQ.

SLM Context Save in L3

Once the new context Save instruction is received, it is treated like a URB read where URB pipeline is used with the new SLM vs URB flag in the request. At the TOP of the Data Pipeline, this flag is used to direct the cycle towards the ways that are populated for SLM. The address will be used which WAY/SET needs to be read same as URB address mapping to physical sets/ways.

SLM Context Restore in L3

The restore request will be treated like a URB write in L3 pipeline where the URB FIFO is used with the new flag to indicate SLM destination. The rest will be similar to Save where the provided address and flag is used to direct the data towards SLM space.

EU Overview

The GEN instruction set is a general-purpose data-parallel instruction set optimized for graphics and media computations. Support for 3D graphics API (Application Programming Interface) Shader instructions is mostly native, meaning that GEN efficiently executes Shader programs. Depending on Shader program operation modes (for example, a Vertex Shader may be executed on a base of a vertex pair, while a Pixel Shader may be executed on a base of a 16-pixel group), translation from 3D graphics API Shader instruction streams into GEN native instructions may be required. In addition, there are many specific capabilities that accelerate media applications. The following feature list summarizes the GEN instruction set architecture:

- SIMD (single instruction multiple data) instructions. The maximum number of data elements per instruction depends on the data type.
- SIMD parallel arithmetic, vector arithmetic, logical, and SIMD control/branch instructions.
- Instruction level variable-width SIMD execution.
- Conditional SIMD execution via destination mask, predication, and execution mask.
- Instruction compaction.
- An instruction may executed in multiple cycles over a SIMD execution pipeline.
- Most GEN instructions have three operands. Some instructions have additional implied source or destination operands. Some instructions have explicit dual destinations.
- Region-based register addressing.
- Direct or indirect (indexed) register addressing.
- Scalar or vector immediate source operand.
- Higher precision accumulator registers are architecturally visible.
- Self-modifying code is not allowed (instruction streams, including instruction caches, are read-only).

CoIssue/Dual Issue:

The EM Pipe has been expanded to handle general integer and all the float operations that FPU pipe can execute as well.

Thread scheduling:

Threads are scheduled with the "oldest first" policy: a thread runs as long as no dependency is encountered. When a switch is required, the oldest thread i.e., the thread which has been spawned the first is the next to execute. After scheduling the next instruction from the currently executing thread, if any of the four units are free, the EU tries to fill them from instructions from other threads (processed in oldest to newest order).

Description:

- Opcodes: All opcodes except DX11 special opcodes.
- Datatype: All datatypes except bytes.
- Accessmode:
  - Align1:
    - No Scattering or Gathering data. This means data in source and destination registers are aligned and packed (data is contiguous in a register).

```
 //Example:
 // allowed, data is contiguous and source and destination regioning map one
to one.
 mov (8) r10.0:f  r11.0<8;8,1>:f

 // not allowed, data from source is strided and requires gathering to write
to destination
 mov (8) r10.0:f  r11.0<4;4,2>:f

 // not allowed, data from source is contiguous but not aligned with
destination.
 //Destination register requires scattering
 mov (8) r10.0<2>:w r11.0<8;8,1>:w

 //not allowed, data from source is contiguous but destination is not
aligned to source
 mov (8) r10.1:f r11.0<4;4,1>:f

 // allowed. Source and destination have stride but are aligned
 mov (4) r10.1:f r11.1<4;4,1>:f
```

    - A scalar on 32b/64b datatype is allowed.
  - Align16
- Addressmode: Direct Addressing
- Register File: GRF/NULL/Immediates. No access to Accumulator.

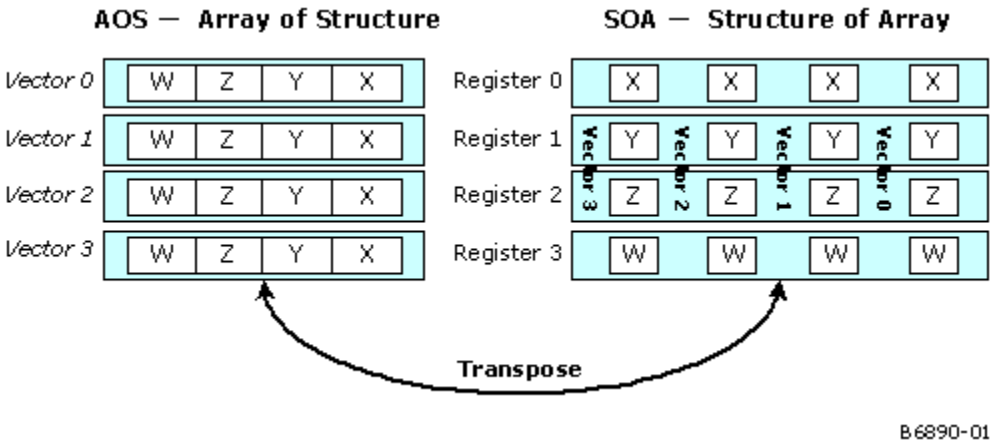
Primary Usage Models

In describing the usage models of the GEN instruction set, the following sections forward reference terminology, syntax, and instructions described later in this specification. For clarity reasons, not all forward references are explained at the point of reference. See the Instruction Set Summary chapter for information about instruction fields and syntax.

AOS and SOA Data Structures

With the Align1 and Align16 access modes, the GEN instruction set provides effective SIMD computation whether data is arranged in array of structures (AOS) form or in structure of arrays (SOA) form. The AOS and SOA data structures are illustrated by the examples in *AOS and SOA Data Structures*. The example shows two different ways of storing four vectors in four SIMD registers. For simplicity, the data vector and the SIMD register both have four data elements. The four data elements in a vector are denoted by X, Y, Z, and W just as for a vertex in 3D geometry. The AOS structure stores one vector in a register and the next vector in another register. The SOA structure stores one data element of each vector in a register and the next element of each vector in the next register and so on. The two structures can be related by a matrix transpose operation.

AOS and SOA Data Structures



GEN 3D and media applications take advantage of such broad architecture support and use both AOS and SOA data arrangements.

- Vertices in 3D Geometry (Vertex Shader and Geometry Shader) are arranged in AOS form and use SIMD4x2 and SIMD4 modes, respectively, as detailed below.
- Pixels in 3D Rasterization (Pixel Shader) are arranged in SOA form and use SIMD8 and SIMD16 modes as detailed below.
- Pixels in media are primarily arranged in SOA form, and occasionally in AOS form with possibly mixed modes of operation that uses region-based addressing extensively.

These are preferred methods; alternative arrangements may also be possible. Shared function resources provide data transpose capability to support both modes of operations: The sampler has a transpose for sample reads, the data port has a transpose for render cache writes, and the URB unit has a transpose for URB writes.

The following 3D graphics API Shader instruction is used in the following sections to illustrate various operation modes:

```
add dst.xyz src0.yxzw src1.zwxy
```

This example is a SIMD instruction that takes two source operands src0 and src1, adds them, and stores the result to the destination operand dst. Each operand contains four floating-point data elements. The data type is determined by the instruction opcode. This instruction also uses source swizzles (.yxzw for src0 and .zwxy for src1) and a destination mask (.xyz). Please refer to the programming specifications of 3D graphics API Shader instructions for more details.

A general register has 256 bits, which can store 8 floating point data elements. For 3D graphics, the mode of operation is (loosely) termed after the data structure as SIMD*mxn*, where *m* is the size of the vector and *n* is the number of concurrent program flows executed in SIMD.

Execution with AOS data structures:

- **SIMD4** (short for SIMD4x1) indicates that a SIMD instruction operates on 4-element vectors storedin registers. There is one program flow.
- **SIMD4x2** indicates that a SIMD instruction operates on a pair of 4-element vectors in registers. There are effectively two programs running side by side with one vector per program.

Execution with SOA data structures, also referred to as “channel serial” execution, mostly uses:

- **SIMD8** (short for SIMD1x8) indicates a SIMD instruction based on the SOA data structure where one register contains one data element (the same one) for each of 8 vectors. Effectively, there are 8 concurrent program flows.
- **SIMD16** (short for SIMD1x16) indicates that a SIMD instruction operates on a pair of registers that contain one data element (the same one) for each of 16 vectors. SIMD16 has 16 concurrent program flows.

SIMD4 Mode of Operation

With a register mapping of src0 to doublewords 0-3 of *r2*, src1 to doublewords 4-7 of *r2* and dst to doublewords 0-3 of *r3*, the example 3D graphics API Shader instruction can be translated into the following GEN instruction:

add (4) r3<4>.xyz:f r2<4>.yzwx:f r2.4<4>.zwxy:f {NoMask}

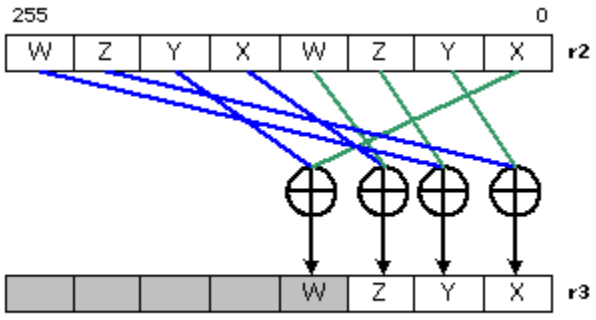
Without diving too much into the syntax definition of a GEN instruction, it is clear that a GEN instruction also takes two source operands and one destination operands. The second term, (4), is the execution size that determines the number of data elements processed by the SIMD instruction. It is similar to the term SIMD Width used in the literature. Each operand is described by the register region parameters such as '<4>' and data type (e.g. "*:f*"). These will be detailed in the SIMD8 Mode of Operation section. The instruction option field, {NoMask}, ensure that the execution occurs for the execution channels shown in the instruction, instead of, possibly, being masked out by the conditional masks of the thread (See Instruction Summary chapter for definition of *MaskCtrl* instruction field).

The operation of this GEN instruction is illustrated in the following figure. In this example, both source operands share the same physical GRF register r2. The two are distinguished by the subregister number. The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 are not modified. In particular, doublewords 4-7 are unchanged as the execution size is 4; doubleword 3 is unchanged due to the destination mask setting.

In this mode of operation, there is only one program flow – any branch decision will be based on a scalar condition and apply to the whole vector of four elements. Option {*NoMask*} ensures that the instruction is not subject to the masks. In fact, most of the instructions in a thread should have {*NoMask*} set.

Even though the execution only performs four parallel add operations, the GEN instruction still executes in 2 cycles (with no useful computation in the second cycle).

A SIMD4 Example



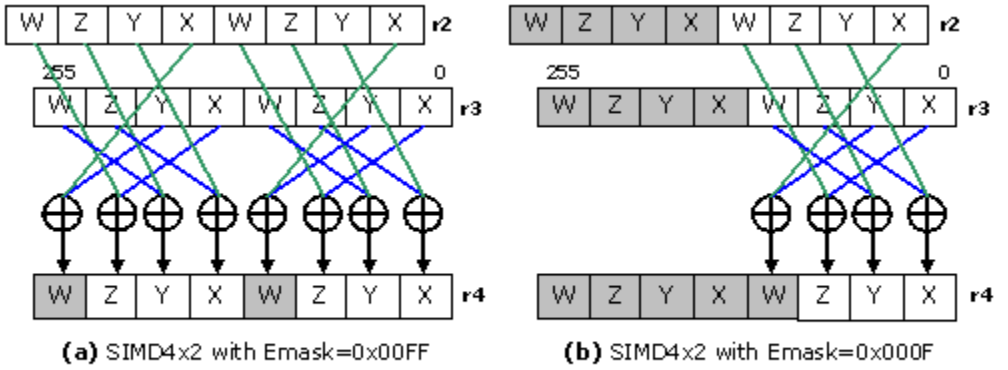
SIMD4x2 Mode of Operation

In this mode, two corresponding vectors from the two program flows fill a GEN register. With a register mapping of src0 to r2, src1 to r3 and dst to r4, the example 3D graphics API Shader instruction can be translated into the following GEN instruction:

add (8) r4<4>.xyz:f r2<4>.yxzw:f r3<4>.zwxy:f

This instruction is subject to the execution mask, which initiated from the dispatch mask. If both program flows are available (e.g. Vertex Shader executed with two active vertices), the dispatch mask is set to 0x00FF. The operation of this GEN instruction is illustrated in *SIMD4x2 Mode of Operation* (a). The source swizzles control the routing of source data elements to the parallel adders corresponding to the destination data elements. The shaded areas in the destination register r3 (doublewords 3 and 7) are unchanged due to the destination mask setting. If only one program flow is available (e.g. the same SIMD4x2 Vertex Shader with only one active vertex), the dispatch mask is set to 0x000F. The operation of the same instruction is shown in *SIMD4x2 Mode of Operation* (b).

SIMD4x2 Examples with Different Emasks

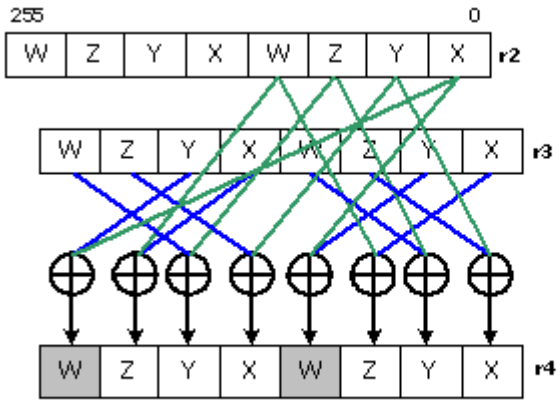


The two source operands only need to be 16-byte aligned, not have to be GRF register aligned. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2, which is shared by the two program flows. The example 3D graphics API Shader instruction can then be translated into the following GEN instruction:

add (8) r4<4>.xyz:f r2<0>.yzwx:f r3<4>.zwxy:f

The only difference here is that the vertical stride of the first source is 0. The operation of this GEN instruction is illustrated in *SIMD4x2 Mode of Operation*.

A SIMD4x2 Example with a Constant Vector Shared by Two Program Flows



SIMD16 Mode of Operation

With 16 concurrent program flows, one element of a vector would take two GRF registers. In this mode, two corresponding vectors from the two program flows fill a GEN register.

With the following register mappings

src0:r2-r9 (with 16 X data elements in r2-r3, Y in r4-5, Z in r6-7 and W in r8-9),

src1:r10-r17,

dst:r18-r25,

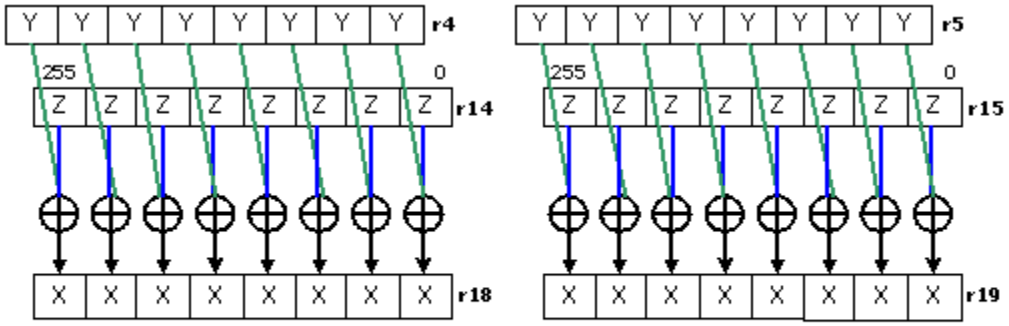
the example 3D graphics API Shader instruction can be translated into the following three GEN instructions:

```
add (16)   r18<1>:f   r4<8;8,1>:f   r14<8;8,1>:f// dst.x = src0.y + src1.z
add (16)   r20<1>:f   r6<8;8,1>:f   r16<8;8,1>:f// dst.y = src0.z + src1.w
add (16)   r22<1>:f   r8<8;8,1>:f   r10<8;8,1>:f  // dst.z = src0.w + src1.x
```

The three GEN instructions correspond to the three enabled destination masks As there is no output for the W elements of dst, no instruction is needed for that element. The first instruction inputs the Y elements of src0 and the Z elements of src1 and outputs the X elements of dst. The operation of this instruction is shown in *SIMD16 Mode of Operation*.

With more than one program flow, the above instructions are also subject to the execution mask. The 16-bit dispatch mask is partitioned into four groups with four bits each. For Pixel Shader generated by the Windower, each 4-bit group corresponds to a 2x2 pixel subspan. If a subspan is not valid for a Pixel Shader instance, the corresponding 4-bit group in the dispatch mask is not set. Therefore, the same instructions can be used independent of the number of available subspans without creating bogus data in the subspans that are not valid.

A SIMD16 Example




```
Add (16) r18<1>:f r4<8;8,1>:f r14<8;8,1>:f {Compr}   // dst.x=src0.y+src1.z
```

B6894-01

Similar to SIMD4x2 mode, a constant may also be shared for the 16 program flows. For example, the first source operand could be a 4-element vector (e.g. a constant) stored in doublewords 0-3 in r2 (AOS format). The example 3D graphics API Shader instruction can then be translated into the following GEN instruction:

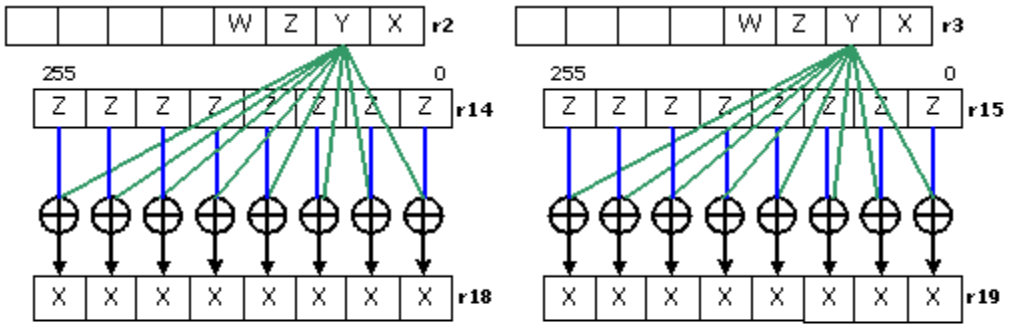
add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr}// dst.x = src0.y + src1.z

add (16) r20<1>:f r2.2<0;1,0>:f r16<8;8,1>:f {Compr}// dst.y = src0.z + src1.w

add (16) r22<1>:f r2.3<0;1,0>:f r10<8;8,1>:f {Compr}// dst.z = src0.w + src1.x

The register region of the first source operand represents a replicated scalar. The operation of the first GEN instruction is illustrated in *SIMD16 Mode of Operation*.

Another SIMD16 Example with an AOS Shared Constant




```
Add (16) r18<1>:f r2.1<0;1,0>:f r14<8;8,1>:f {Compr}   // dst.x=src0.y+src1.z
```

B6895-01

SIMD8 Mode of Operation

Each compressed instruction has two corresponding native instructions. Taking the example instruction shown in *SIMD16 Mode of Operation*, it is equivalent to the following two instructions.

```
add (8)   r18<1>:f   r4<8;8,1>:f   r14<8;8,1>:f // dst.x[7:0] = src0.y + src1.z
add (8)   r19<1>:f   r5<8;8,1>:f   r15<8;8,1>:f   {SecHalf}// dst.x[15:8] = src0.y + src1.z
```

Therefore, SIMD8 can be viewed as a special case for SIMD16.

There are other reasons that SIMD8 instructions may be used. Within a program with 16 concurrent program flows, some time SIMD8 instruction must be used due to architecture restrictions. For example, the address register a0 only have 8 elements, if an indirect GRF addressing is used, SIMD16 instructions are not allowed.

Messages

Communication between the EUs and the shared functions and between the fixed function pipelines (which are not considered part of the "Subsystem") and the EUs is accomplished via packets of information called *messages*. Message transmission is requested via the send instruction. Refer to the send instruction definition in the *ISA Reference* chapter for details.

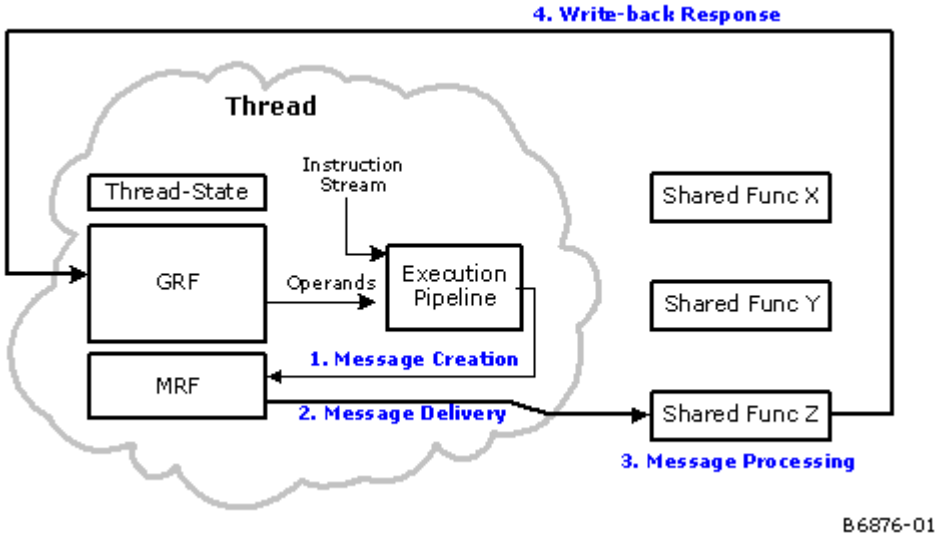
The information transmitted in a message falls into two categories:

- **Message Payload.**
- Associated ("sideband") information provided by:
  - **Message Descriptor.** Specified with the send instruction. Included in the message descriptor is control and routing information such as the target function ID, message payload length, response length, etc.
  - Additional information provided by the send instruction, e.g., the starting destination register number, the execution mask (EMASK), etc.
  - A small subset of Thread State, such as the Thread ID, EUID, etc.

The software view of messages is shown in *Messages* . There are four basic phases to a message's lifetime as illustrated below:

1. **Creation.**
2. **Delivery.** The thread issues the message for delivery via the send instruction. The send instruction also specifies the destination shared function ID (SFID), and where in the GRF any response is to be directed. The messaging subsystem enqueues the message for delivery and eventually routes the message to the specified shared function.
3. **Processing.** The shared function receives the message and services it accordingly, as defined by the shared function definition.
4. **Writeback.** If called for, the shared function delivers an integral number of registers of data to the thread's GRF in response to the message.

Data Flow Associated With Messages



Message Payload Containing a Header

For most shared functions, the first register of the message payload contains the *header payload* of the message (or simply the *message header*). Consequently, the rest of the message payload is referred to as the *data payload*.

Messages to Extended Math do not have a header and only contain data payload. Those messages may be referred to as header-less messages. Messages to Gateway combine the header and data payloads in a single message register.

Writebacks

Some messages generate return data as dictated by the 'function-control' (opcode) field of the 'send' instruction (part of the <desc> field). The Gen4 execution unit and message passing infrastructure do not interpret this field in any way to determine if writeback data is to be expected. Instead explicit fields in the 'send' instruction to the execution unit the starting GRF register and count of returning data. The execution unit uses this information to set in-flight bits on those registers to prevent execution of any instruction which uses them as an operand until the register(s) is(are) eventually written in response to the message. If a message is not expected to return data, the 'send' instruction's writeback destination specifier (<post_dest>) must be set to 'null' and the response length field of <desc> must be 0 (see 'send' instruction for more details).

The writeback data, if called for, arrives as a series of register writes to the GRF at the location specified by the starting GRF register and length as specified in the 'send' instruction. As each register is written back to the GRF, its in-flight flag is cleared and it becomes available for use as an instruction operand. If a thread was suspended pending return of that register, the dependency is lifted and the thread is allowed to continue execution (assuming no other dependency for that thread remains outstanding).

Message Delivery Ordering Rules

All messages between a thread and an individual shared function are delivered in the ordered they were sent. Messages to different shared functions originating from a single thread may arrive at their respective shared functions out of order.

The writebacks of various messages from the shared functions may return in any order. Further individual destination registers resulting from a single message may return out of order, potentially allowing execution to continue before the entire response has returned (depending on the dependency chain inherent in the thread).

Execution Mask and Messages

The Gen4 Architecture defines an Execution Mask (EMask) for each instruction issued. This 16b bit-field identifies which SIMD computation channels are enabled for that instruction. Since the ‘send’ instruction is inherently scalar, the EMask is ignored as far as instruction dispatch is concerned. Further the execution size has no impact on the size of the ‘send' instruction’s implicit move (it is always 1 register regardless of specified execution size).

The 16b EMask is forwarded with the message to the destination shared function to indicate which SIMD channels were enabled at the time of the ‘send’. A shared function may interpret or ignore this field as dictated by the functionality it exposes. For instance, the Extended Math shared function observes this field and performs the specified operation only on the operands with enabled channels, while the DataPort writes to the render cache ignore this field completely, instead using the pixel mask included in-band in the message payload to indicate which channels carry valid data.

End-Of-Thread (EOT) Message

The final instruction of all threads must be a *send* instruction that signals 'End-Of-Thread' (EOT). An EOT message is one in which the EOT bit is set in the *send* instruction's 32b <desc> field. When issuing instructions, the EU looks for an EOT message, and when issued, shuts down the thread from further execution and considers the thread completed.

Only a subset of the shared functions can be specified as the target function of an EOT message, as shown in the table below.

Target Shared Functions supporting EOT messages	Target Shared Functions not supporting EOT messages
Null, DataPortWrite, URB, MessageGateway, ThreadSpawner	DataPortRead, Sampler

Both the fixed-functions and the thread dispatcher require EOT notification at the completion of each thread. The thread dispatcher and fixed functions in the 3D pipeline obtain EOT notification by snooping all message transmissions, regardless of the explicit destination, looking for messages which signal end-of-thread. The Thread Spawner in the media pipeline does not snoop for EOT. As it is also a shared function, all threads generated by Thread Spawner must send a message to Thread Spawner to explicity signal end-of-thread.

The thread dispatcher, upon detecting an end-of-thread message, updates its accounting of resource usage by that thread, and is free to issue a new thread to take the place of the ended thread. Fixed functions require end-of-thread notification to maintain accounting as to which threads it issued have completed and which remain outstanding, and their associated resources such as URB handles.

Unlike the thread dispatcher, fixed-functions discriminate end-of-thread messages, only acting upon those from threads which they originated, as indicated by the 4b fixed-function ID present in R0 of end-of-thread message payload. This 4b field is attached to the thread at new-thread dispatch time and is placed in its designated field in the R0 contents delivered to the GRF. Thus to satisfy the inclusion of the fixed-function ID, the typical end-of-thread message generally supplies R0 from the GRF as the first register of an end-of-thread message.

As an optimization, an end-of-thread message may be overload upon another "productive" message, saving the cost in execution and bandwidth of a dedicated end-of-thread message. Outside of the end-of-thread message, most threads issue a message just prior to their termination (for instance, a Dataport write to the framebuffer) so the overloaded end-of-thread is the common case. The requirement is that the message contains R0 from the GRF (to supply the fixed-function ID), and that destination shared function be either (a) the URB; (b) the Read or Write Dataport; or, (c) the Gateway, as these functions reside on the O-Bus. In the case where the last real message of a thread is to some other shared function, the thread must issue a separate message for the purposes of signaling end-of-thread to the "null" shared function.

Performance

The Gen4 Architecture imposes no requirement as to a shared function's latency or throughput. Due to this as well as factors such as message queuing, shared bus arbitration, implementation choices in bus bandwidth, and instantaneous demand for that function, the latency in delivering and obtaining a response to a message is non-deterministic. It is expected that a Gen4 implementation has some notion of fairness in transmission and servicing of messages so as to keep latency outliers to a minimum.

Other factors to consider with regard to performance:

Description
Software prefetching techniques may be beneficial for long latency data fetches (i.e. issue a load early in the thread for data that is required late in the thread).

Message Description Syntax

All message formats are defined in terms of DWords (32 bits). The message registers in all cases are 256 bits wide, or 8 DWords. The registers and DWords within the registers are named as follows, where n is the register number, and d is the DWord number from 0 to 7, from the least significant DWord at bits [31:0] within the 256-bit register to the most significant DWord at bits [255:224], respectively. For writeback messages, the register number indicates the offset from the specified starting destination register.

Dispatch Messages: **R**n.d

Dispatch messages are sent by the fixed functions to dispatch threads. See the fixed function chapters in the *3D and Media* volume.

SEND Instruction Messages: **M**n.d

These are the messages initiated by the thread via the SEND instruction to access shared functions. See the chapters on the shared functions later in this volume.

Writeback Messages: **W**n.d

These messages return data from the shared function to the GRF where it can be accessed by thread that initiated the message.

The bits within each DWord are given in the second column in each table.

Message Errors

Messages are constructed via software, and not all possible bit encodings are legal, thus there is the possibility that a message may be sent containing one or more errors in its descriptor or payload contents. There are two points of error detection in the message passing system: (a) the message delivery subsystem is capable of detecting bad FunctionIDs and some cases of bad message lengths; (b) the shared functions contain various error detection mechanisms which identify bad sub-function codes, bad message lengths, and other misc errors. The error detection capabilities are specific to each shared function. The execution unit hardware itself does not perform message validation prior to transmission.

In both cases, information regarding the erroneous message is captured and made visible through MMIO registers, and the driver notified via an interrupt mechanism.

The set of possible errors is listed in *Message Errors* with the associated outcome.

Error Cases

Error	Outcome
Bad Shared Function ID	The message is discarded before reaching any shared function. If the message specified a destination, those registers will be marked as in-flight, and any future usage by the thread of those registers will cause a dependency which will never clear, resulting in a hung thread and eventual time-out.
Unknown opcode Incorrect message length	The destination shared function detects unknown opcodes (as specified in the 'send' instructions <desc> field), and known opcodes where the message payload is either too long or too short, and threats these cases as errors. When detected, the shared function latches and makes available via MMIO registers the following information: the EU and thread ID which sent the message, the length of the message and expected response, and any relevant portions of the first register (R0) of the message payload. The shared function alerts the driver of an erroneous message through and interrupt mechanism then continues normal operation with the subsequent message.
Bad message contents in payload	Detection of bad data is an implementation decision of the shared function. Not all fields may be checked by the shared function, so an erroneous payload may return bogus data or no data at all. If an erroneous value is detected by the shared function, it is free to discard the message and continue with the subsequent message. If the thread was expecting a response, the destination registers specified in the associated 'send' instruction are never cleared potentially resulting in a hung thread and time-out.
Incorrect response length	Case: too few registers specified – the thread may proceed with execution prior to all the data returning from the shared function, resulting in the thread operating on bad data in the GRF. Case: too many registers specified – the message response does not clear all the registers of the destination. In this case, if the thread references any of the residual registers, it may hand and result in an eventual time-out.
Improper use of End-Of-Thread (EOT)	Any 'send' instruction which specifies EOT must have a 'null' destination register. The EU enforces this and, if detected, will not issue the 'send' instruction, resulting in a hung

Error	Outcome
	thread and an eventual time-out. The 'send' instruction specifies that EOT is only recognized if the <desc> field of the instruction is an immediate. Should a thread attempt to end a thread using a <desc> sourced from a register, the EOT bit will not be recognized. In this case, the thread will continue to execute beyond the intended end of thread, resulting in a wide range of error conditions.
Two outstanding messages using overlapping GRF destinations ranges	This is not checked by HW. Due to varying latencies between two messages, and out-of-order, non-contiguous writeback cycles, the outcome in the GRF is indeterminate; may be the result from the first message, or the result from the second message, or a combination of both.

Registers and Register Regions

This topic is currently under development.

Register Files

GEN registers are grouped into different name spaces called register files. There are two register files, the General Register File and the Architecture Register File. A third encoding of some register file instruction fields indicates immediate operands within instructions rather than a register file.

- General Register File (GRF): The GRF contains general-purpose read-write registers.
- Architecture Register File (ARF): The ARF contains all architectural registers defined for specific purposes, including address registers (*a#*), accumulators (*acc#*), flags (*f#*), notification count (*n#*), instruction pointer (*ip*), null register (*null*), etc.
- Immediate: Certain instructions can take immediate source operands. A distinct register file field encoding indicates an immediate operand.

Each thread executed in an EU has its own thread context, a dedicated register space that is not shared between threads, whether executing on a common EU or on a different EU. In the rest of the chapters in this volume, register access is relative to a given thread.

GRF Registers

Number of Registers: Various
Default Value: None
Normal Access: RW
Elements: Various
Element Size: Various
Element Type: Various
Access Granularity: Byte
Write Mask Granularity: Byte
Indexable? Yes

Registers in the General Register File are the most commonly used read-write registers. During the execution of a thread, GRF registers are used to store the temporary data, and serve as the destination to receive data from shared function units (and some times from a fixed function unit). They are also used to store the input (initialization) data when a thread is created. By allowing fixed function hardware to initialize some portion of GRF registers during thread dispatch time, GEN architecture can achieve better parallelism. A thread's execution efficiency can also be improved as some data are already in the register to be executed upon. Besides these registers containing thread's payload, the rest of GRF registers of a thread are not initialized.

Summary of GRF Registers

Register File	Register Name	Description
General Register File (GRF)	r#	General purpose read write registers

Each execution unit has a fixed size physical GRF register RAM. The GRF register RAM is shared by all threads on the EU. Each thread has a dedicated space of 128 register, r0 through r127.

GRF registers can be accessed using region-based addressing at byte granularity (both read and write). A source operand must be contained within two adjacent registers. A destination operand must be contained within one register. GRF registers support direct addressing and register-indirect addressing. Register-indirect addressing uses the address registers (ARF registers a#) and an immediate address offset value.

When accessing (read and/or write) outside the GRF register range allocated for a given thread either through direct or indirect addressing, the result is unpredictable.

Register Size	Allocation Granularity	Number per Thread
256 bits	Fixed allocation of 128 registers	128 registers

ARF Registers

This topic is currently under development.

ARF Registers Overview

Besides registers that are directly indicated by a unique register file coding, all other registers belong to the Architecture Register File (ARF). Encodings of architecture register types are based on the MSBs of the register number field, RegNum, in the instruction word. The RegNum field has 8 bits. The 4 MSBs, RegNum[7:4], represent the architecture register type. This is summarized in the *Summary of Architecture Registers* table below.

Description
GRF registers are directly indicated by a unique register file encoding.

Summary of Architecture Registers

Register Type (RegNum [7:4])	Register Name	Register Count	Description
0000b	*null*	1	Null register
0001b	*a0.#*	1	Address register
0010b	*acc#*	10	Accumulator register
0011b	*f#.#*	2	Flag register
0100b	*ce#*	1	Channel Enable register
0101b	*msg#*	32	Message Control Register
0110b	*sp*	1	Stack Pointer Register
0111b	*sr0.#*	1	State register
1000b	*cr0.#*	1	Control register
1001b	*n#*	2	Notification Count register
1010b	*ip*	1	Instruction Pointer register
1011b	*tdr*	1	Thread Dependency register
1100b	*tm0*	2	TimeStamp register
1101b	*fc#.#*	39	Flow Control register
1110b	*Reserved*		Reserved

Programming Note	
Context:	ARF Registers Overview
The remaining register number field RegNum[3:0] is used to identify the register number of a given architecture register type. Therefore, the maximum number of registers for a given architecture register type is limited to 16. The subregister number field, SubRegNum, in the instruction word has 5 bits. It is used to address subregister regions for an architecture register supporting register subdivision. The SubRegNum field is in units of bytes. Therefore, the maximum number of bytes of an architecture register is limited to 32. Depending on the alignment restriction of a register type, only certain encodings of SubRegNum field apply for an architecture register. The detailed definitions are provided in subsequent sections.	

Programming Note	
Context:	ARF Registers Overview
In general an ARF register can be dst (destination) or src0 (source 0, first source operand) for an instruction. Depending on the register and the instruction, other restrictions may apply.	

Access Granularity

ARF registers may be accessed with subregister granularity according to the descriptions below and following the same rule of region-based addressing for GRF.

The machine code for register number and subregister number of ARF follows the same rule as for other register files with byte granularity. For an ARF as a source operand, the region-based address controls the source swizzle mux. The destination subregister number and destination horizontal stride can be used to generate the destination write mask at byte level.

Subregister fields of an ARF register may not all be populated (indicated by the subregister being indicated as reserved). Writes to unpopulated subregisters are dropped; there are no side effect. Reads from unpopulated subregisters, if not specified, return unpredictable data.

Some ARF registers are read-only. Writes to read-only ARF registers are dropped and there are no side effects.

Null Register

Null Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0000b
Number of Registers:	1
Default Value:	N/A
Normal Access:	N/A
Elements:	N/A
Element Size:	N/A
Element Type:	N/A
Access Granularity:	N/A
Write Mask Granularity:	N/A
SecHalf Control?	N/A
Indexable?	No

The null register is a special encoding for an operand that does not have a physical mapping. It is primarily used in instructions to indicate non-existent operands. Writing to the null register has no side effect. Reading from the null register returns an undefined result.

The null register can be used where a source operand is absent. For example, for a single source operand instruction such as MOV or NOT, the second source operand src1 must be a null register.

When the null register is used as the destination operand of an instruction, it indicates the computed results are not stored in any registers. However, implied writes to the accumulator register, if applicable, may still occur for the instruction. When the conditional modifier is present, updates to the selected flag register also occur. In this case, the register region fields of the 'null' operand are valid.

Another example use is to use the null register as the posted destination of a *send* instruction for data write to indicate that no write completion acknowledgement is required. In this case, however, the register region fields are still valid. The null register can also be the first source operand for a send instruction indicating the absent of the implied move. See the *send* instruction for details.

Address Register

Address Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0001b
Number of Registers:	1
Default Value:	None
Normal Access:	RW
Elements:	16
Element Size:	16 bits
Element Type:	UW or UD
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	N/A
Indexable?	No

Description
There are sixteen address subregisters forming a 16-element vector. Each address subregister contains 16 bits. Address subregisters can be used as regular source and destination operands, as the indexing addresses for register-indirect-addressed access of GRF registers, and also as the source of the message descriptor for the *send* instruction.

Register and Subregister Numbers for Address Register

RegNum[3:0]	SubRegNum[4:0]
0000b = a0 All other encodings are reserved.	When register a0 or subregisters in a0 are used as the address register for register-indirect addressing, the address subregisters must be accessed as unsigned word integers. Therefore, the subregister number field must also be word-aligned. 00000b = **a0.0:uw** 00010b = **a0.1:uw** 00100b = **a0.2:uw** 00110b = **a0.3:uw** 01000b = **a0.4:uw** 01010b = **a0.5:uw** 01100b = **a0.6:uw** 01110b = **a0.7:uw** 10000b = **a0.8:uw** 10010b = **a0.9:uw** 10100b = **a0.10:uw** 10110b = **a0.11:uw** 11000b = **a0.12:uw** 11010b = **a0.13:uw** 11100b = **a0.14:uw** 11110b = **a0.15:uw** All other encodings are reserved. However, when register a0 or subregisters in a0 are explicit source and/or destination registers, other data types are allowed as long as the register region falls in the 128-bit range.

Address Register Fields

DWord	Bits	Description
7	31:16	**Address subregister a0.15:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.14:uw**. Follows the same format as **a0.2**.
6	31:16	**Address subregister a0.13:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.12:uw**. Follows the same format as **a0.2**.
5	31:16	**Address subregister a0.11:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.10:uw**. Follows the same format as **a0.2**.
4	31:16	**Address subregister a0.9:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.8:uw**. Follows the same format as **a0.2**.
3	31:16	**Address subregister a0.7:uw**. Follows the same format as **a0.3**.
	15:0	**Address subregister a0.6:uw**. Follows the same format as **a0.2**.
2	31:16	**Address subregister a0.5:uw**. Follows the same format as a0.3.
	15:0	**Address subregister a0.4:uw**. Follows the same format as a0.2.
1	31:16	**Address subregister a0.3:uw**. This field, with only the lower 12 bits populated can be used as an unsigned integer for register-indirect register addressing. Format: U12
	15:0	**Address subregister a0.2:uw**. This field, with only the lower 12 bits populated can be used as an unsigned integer for register-indirect register addressing. Format: U12
0	31:16	**Address subregister a0.1:uw**. This field can be used for register-indirect register addressing or serve as message descriptor for a *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For a *send* instruction, it provides the higher 16 bits of <desc>. Format: U12 or U16.
	15:0	**Address subregister a0.0:uw**. This field can be used for register-indirect register addressing or serve as message descriptor for a *send* instruction. When used for register-indirect register addressing, it is a 12-bit unsigned integer. For a *send* instruction, it provides the lower 16 bits of <desc>. Format: U12 or U16.

When used as a source or destination operand, the address subregisters can be accessed individually or as a group. In the following example, the first instruction moves 8 address subregisters to the first half of GRF register r1, the second instruction replicates a0.4:uw as an unsigned word to the second half of r1, the third instruction moves the first 4 words in r1 into the first 4 address subregisters, and the fourth instruction replicates r1.4 as a unsigned word to the next 4 address subregisters.

```
mov (8) r1.0<1>:uw a0.0<8;8,1>:uw // r1.n = a0.n for n = 0 to 7 in words
mov (8) r1.8<1>:uw a0.4<0;1,0>:uw // r1.m = a0.4 for m = 8 to 15 in words
mov (4) a0.0<1>:uw r1.0<4;4,1>:uw // a0.n = r1.n for n = 0 to 3 in words
mov (4) a0.4<1>:uw r1.4<0;1,0>:uw // a0.m = r1.4 for m = 4 to 7 in words
```

When used as the register-indirect addressing for GRF registers, the address subregisters can be accessed individually or as a group. When accessed as a group, the address subregisters must be group-aligned. For example, when two address subregisters are used for register indirect addressing, they must be aligned to even address subregisters. In the following example, the first instruction is legal. However, the second one is not. As ExecSize = 8 and the width of src0 is 4, two address subregisters are used as row indices, each pointing to 4 data elements spaced by HorzStride = 1 dword. For the first instruction, the two address subregisters are a0.2:uw and a0.3:uw. The two align to a DWord group in the address register. However, the two address subregisters for the second instruction are a0.3:uw and a0.4:uw. They are not DWord-aligned in the address register and therefore violate the above mentioned alignment rule.

```
mov (8) r1.0<1>:d r[a0.2]<4,1>:d // a0.2 and a0.3 are used for src1
mov (8) r1.0<1>:d r[a0.3]<4,1>:d // ILLEGAL use of register indirect
```

Implementation restriction: When used as the source operand <desc> for the send instruction, only the first dword subregister of a0 register is allowed (i.e. a0.0:ud, which can be viewed as the combination of a0.0:uw and a0.1:uw). In addition, it must be of UD type and in the following form <desc> = a0.0<0;1,0>:ud.

Restriction: Elements a0.0 and a0.1 have 16 bits each, but the rest of the elements (a0.2:uw through a0.7:uw) only have 12 bits populated each. 12-bit precision supports full indirect-addressing capability for the largest GRF register range. Software must observe the asymmetry of the implementation. When a0.0:uw and a0.1:uw are the source or destination, full 16-bit precision is preserved. However, when a0.2:uw to a0.7:uw are the destination, the high 4 bits for each element are dropped; when they are the source, hardware inserts zero to the high 4 bits for each element.

Performance Note: There is only one scoreboard for the whole address register. When a write to some subregisters is in flight, hardware stalls any instruction writing to other subregisters. Software may use the destination dependency control { NoDDClr} to improve performance in this case. Similarly, when a write to some subregisters is in flight, hardware stalls any instruction sourcing other subregisters until the write retires.

Accumulator Registers

Accumulator Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0010b
Number of Registers:	10
Default Value:	None
Normal Access:	RW

Accumulator registers can be accessed either as explicit or implied source and/or destination registers. To a programmer, each accumulator register may contain either 8 DWords or 16 Words of data elements. However, as described in the Implementation Precision Restriction notes below, each data element may have higher precision with added guard bits not indicated by the numeric data type.

Accumulator capabilities vary by data type, not just data size, as described in the Accumulator Channel Precision table below. For example, D and F are both 32-bit data types, but differ in accumulator support.

See the Accumulator Restrictions section for information about additional general accumulator restrictions and also accumulator restrictions for specific instructions.

Accumulator Registers
There are 10 accumulator registers. The accumulator registers are of two types.

Register and Subregister Numbers for Accumulator Registers

RegNum[3:0]	SubRegNum[4:0]
0000b-1001b = acc0-acc9 All other encodings are reserved.	Reserved: MBZ

- Accumulators are updated implicitly only if the AccWrCtrl bit is set in the instruction. The Accumulator Disable bit in control register cr0.0 allows software to disable the use of AccWrCtrl for implicit accumulator updates. The write enable in word granularity for the instruction is used to update the accumulator. Data in disabled channels is not updated.
- When an accumulator register is an implicit source or destination operand, hardware always uses acc0 by default and also uses acc1 if the execution size exceeds the number of elements in acc0. When implicit access to acc1 is required, QtrCtrl is used. Note that QtrCtrl can be used only if acc1 is accessible for a given data type. If acc1 is not accessible for a given data type, QtrCtrl defaults to acc0.

Description
acc0 and acc1 are supported for half-precision (HF, Half Float) and single-precision (F, Float). Use QtrCtrl of Q2 or Q4 to access acc1 for Float. use QtrCtrl of H2 to access acc1 for Half Float.

Examples:

```
// Updates acc0 and acc1 because it is SIMD16:
add (16) r10:f r11:f r12:f {AccWrEn}
// Updates acc0 because it is SIMD8:
add (8) r10:f r11:f r12:f {AccWrEn}
// Updates acc1. Implicit access to acc1 using QtrCtrl:
add (8) r10:f r11:f r12:f {AccWrEn, Q2}
// Updates acc1 for Half Floats using QtrCtrl:
add (16) r10:hf r11:hf r12:hf {AccWrEn, H2}
```

- It is illegal to specify different accumulator registers for source and destination operands in an instruction (e.g. "*add (8) acc1:f acc0:f*"). The result of such an instruction is unpredictable.

Limits on SIMD16 Float Operations
Accumulator registers may be accessed explicitly as src0 operands only.

- Swizzling is not allowed when an accumulator is used as an implicit source or an explicit source in an instruction.
- Reading accumulator content with a datatype different from the previous write will result in undeterministic values.
- Word datatype write to accumulator is not allowed when destination is odd offset strided by 2.
- For any DWord operation, including DWord multiply, accumulator can store up to 8 channels of data, with only acc0 supported.
- When an accumulator register is an explicit destination, it follows the rules of a destination register. If an accumulator is an explicit source operand, its register region must match that of the destination register with the exception(s) described below.

Exceptions
When HalfFloats access accumulator, interleaved reads/writes is not allowed. This means, accumulator must be accessed in pairs of half-floats.

When OWords of accumulators are accessed, the source and destination OWords may be different. For example, the following instructions are allowed:
`mov (4) r10.4<1>:f acc0.0<1>:f`
`add (4) r10.0<1>:f acc0.4<1>:f r11.0<1>:f`
`mov (8) r10.8<1>:uw acc0.0<1>:uw`
`add (8) r10.0<1>:uw acc0.8<1>:uw r11.0<1>:uw`
The source and destination datatypes MUST be the same for such access of accumulator.
If destination is contained within one register, source must also be contained within one accumulator register.

Exceptions

When source and destination datatypes are mixed between floats and half-floats, the source and destination Owords may be different. A half float source can be aligned to either the lower or the upper word. *Refer to Special Case Handling of Mixed Mode datatypes in Section: Register Region Restrictions.*
`mul (4) r8.0<1>:f acc0.1<2>:hf r7<1>:f`
`mov (4) r8.0<1>:f acc0.1<2>:hf`

Half floats can be written to either the lower or the upper word of the accumulator. However, this is not supported for integer word operations.
Accumulator should not be used as destination for double-float to float format conversion.
When HalfFloats access accumulator, use of Q3/Q4 quarter control is not allowed.

Implementation Precision Restriction: As there are only 64 bits per channel in DWord mode (D and UD), it is sufficient to store the multiplication result of two DWord operands as long as the post source modified sources are still within 32 bits. If any one source is type UD and is negated, the negated result becomes 33 bits. The DWord multiplication result is then 65 bits, bigger than the storage capacity of accumulators. Consequently, the results are unpredictable.

Implementation Precision Restriction: As there are only 33 bits per channel in Word mode (W and UW), it is sufficient to store the multiplication result of two Word operands with and without source modifier as the result is up to 33 bits. Integers are stored in accumulator in 2's complement form with bit 32 as the sign bit. As there is no guard bit left, the accumulator can only be sourced once before running into a risk of overflowing. When overflow occurs, only modular addition can generate a correct result. But in this case, conditional flags may be incorrect. When saturation is used, the output is unpredictable. This is also true for other operations that may result in more than 33 bits of data. For example, adding UD (FFFFFFFFh) with D (FFFFFFFFh) results in 1FFFFFFFEh. The sign bit is now at bit 34 and is lost when stored in the accumulator. When it is read out later from the accumulator, it becomes a negative number as bit 32 now becomes the sign bit.

Accumulator Channel Precision

Data Type	Accumulator Number	Number of Channels	Bits Per Channel	Description
DF	acc0	4	64	When accumulator is used for Double Float, it has the exact same precision as any GRF register.
DF	N/A	N/A	N/A	Accumulator is not supported for this data

Data Type	Accumulator Number	Number of Channels	Bits Per Channel	Description
				type.
F	acc0/acc1	8	32	When accumulator is used for Float, it has the exact same precision as any GRF register.
HF	N/A	N/A	N/A	Not supported data type.
HF	acc0/acc1	16	16	When accumulator is used for Half Float, it has the exact same precision as any GRF register.
Q	N/A	N/A	N/A	Not supported data type.
D (UD)	acc0	8	33/64	When the internal execution data type is doubleword integer, each accumulator register contains 8 channels of (extended) doubleword integer values. The data are always stored in accumulator in 2's complement form with 64 bits total regardless of the source data type. This is sufficient to construct the 64-bit D or UD multiplication results using an instruction macro sequence consisting of *mul*, *mach*, and *shr* (or *mov*).
W (UW)	acc0	16	33	When the internal execution data type is word integer, each accumulator register contains 16 channels of (extended) word integer values. The data are always stored in accumulator in 2's complement form with 33 bits total. This supports single instruction multiplication of two word sources in W and/or UW format.
B (UB)	N/A	N/A	N/A	Not supported data type.

Accumulators
Accumulators acc2-acc9

These are accumulator registers defined for a special purpose. They are used to emulate IEEE-compliant fdiv and sqrt macro operations. The access is different from acc0 and acc1. Each of these accumulator registers are defined as 256-bit registers having 8 DWords. These may be accessed explicitly or implicitly.

- These registers may be accessed explicitly only by a *mov* operation, with no source modifiers, condition modifiers, or saturation. When accessed explicitly, the datatype must be D. On reads, the low 2 bits of each DWord are valid data. The other bits are undefined. On writes, the low two bits are updated and other bits are dropped.

  Example:

```
// Move 256 bits from acc2 to r10. Just low two bits of each DWord are valid:

mov (8) r10:ud acc2:ud

// Move 256 bits from r10 to acc2. Just low two bits of each DWord are updated:
mov (8) acc2:ud r10:ud
```

- These registers are accessed implicitly by three opcodes defined for the macro operations. **Note:** These macro operations are defined under the *math* opcode section. The macro descriptions also define the restrictive implicit uses of these registers.
- Implicit access across accumulator registers is required for each source operand for these macro instructions. These opcodes are accessed in Align16 mode only. The Channel Select bits in the instruction are used to implicitly address the different accumulators for each source. Similarly the Channel Enable bits are used to implicitly address the accumulators for destination. The noacc value is specified when no write to accumulator is required; think of it as a null.

Channel Select/Channel Enable Encoding for Implicit Source/Destination Access

Encoding	Accumulator Register
00000000b	acc2
00000001b	acc3
00000010b	acc4
00000011b	acc5
00000100b	acc6
00000101b	acc7
00000110b	acc8
00000111b	acc9
00001000b	noacc

Project:	CHV, BSW

On a context save, the sequence of operations shown below MUST be used to save acc2-acc9 registers.

```
mov(8) r113:ud acc2:ud      { NoMask } //acc2
 mov(8) r114:ud acc2.yx:ud  {Align16}    //acc3
 mov(8) r115:ud acc2.zx:ud  {Align16}    //acc4
 mov(8) r116:ud acc2.wx:ud {Align16}     //acc5

mov(8) r113:ud acc2.xy:ud  {Align16}    //acc6

mov(8) r114:ud acc2.yy:ud    {Align16}  //acc7

mov(8) r115:ud acc2.zy:ud  {Align16}   //acc8
 mov(8) r116:ud acc2.wy:ud {Align16}    //acc9
```

On a context restore, the sequence of operations shown below MUST be used to restore acc2-acc9 registers.

```
mov(8) acc2:ud r100:ud      { NoMask } //acc2

mov(8) acc2.x:ud r101:ud    {Align16}    //acc3
```

Project:	CHV, BSW

```
mov(8) acc2.y:ud r102:ud    {Align16}    //acc4
mov(8) acc2.xy:ud r103:ud   {Align16}    //acc5

mov(8) acc2.z:ud r100:ud    {Align16}    //acc6
mov(8) acc2.xz:ud r101:ud   {Align16}    //acc7
mov(8) acc2.yz:ud r102:ud   {Align16}    //acc8
mov(8) acc2:ud r103:ud      {Align16}    //acc9
```

Flag Register

Flag Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0011b
Number of Registers:	2
Default Value:	None
Normal Access:	RW
Elements:	2
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	Yes
Indexable?	No

Description
There are two flag registers, f0 and f1.

Each flag register contains two 16-bit subregisters. Each flag bit corresponds to a data channel. Predication uses flag values to enable or disable channels. Conditional modifiers assign flag values. If an instruction uses both predication and conditional modifiers, both features use the same flag register or subregisters.

Description
Flags can be split to halfs, quarters, or eighths using the QtrCtrl and NibCtrl instruction fields. Those fields affect the selection of flags for predication and conditional modifiers, but do not affect reading or writing flags as explicit instruction operands.

The values held in the individual bits of a flag register are the result of the most recent instruction with a conditional modifier and specifying that flag register. For example:

```
add.nz.f0.0 ...
```

Updates flag subregister f0.0 with the per-channel results of the not zero condition.

The flag register has per-bit write enables. When being updated as the secondary destination associated with a conditional modifier, only the bits corresponding to the enabled channels in *EMask* are updated. Other bits in the flag subregister are unchanged.

Flag registers and subregisters can also be explicit source or destination operands.

The *sel* instruction does not update flags.

Register and Subregister Numbers for Flag Register

RegNum[3:0]	SubRegNum[4:0]
0000b = f0:ud 0001b = f1:ud Other encodings are reserved.	00000b = f*n*.0:uw 00010b = f*n*.1:uw Other encodings are reserved.

Project	Description
CHV, BSW	Reference an entire flag register as f0:ud or f1:ud. Reference the flag subregisters as f0.0:uw, f0.1:uw, f1.0:uw, and f1.1:uw.

Channel Enable Register

Channel Enable Register Summary [CHV, BSW]

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0100b
Number of Registers:	1
Default Value:	N/A
Normal Access:	RO
Elements:	1
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	N/A
SecHalf Control?	No
Indexable?	No

Register and Subregister Numbers for Channel Enable Register [CHV, BSW]

RegNum[3:0]	SubRegNum[4:0]
0000b = ce All other encodings are reserved.	00000b = **ce:ud**. All other encodings are reserved.

Channel Enable Register Fields [CHV, BSW]

DWord	Bits	Description
0	31:0	**Channel Enable Register ce0.0:ud** Format: U32 This field contains 32 bits of Channel Enables for the current instruction.

Message Control Registers

Message Control Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0101b
Number of Registers:	8
Default Value:	None
Normal Access:	RW
Elements:	2
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Word
Write Mask Granularity:	Word

Register and SubRegister Numbers for Message Control Registers

RegNum[3:0]	SubRegNum[4:0]
0000b - 0111 = msg0 - msg7 All other encodings are reserved.	MBZ

These are specific control registers used to track messaging. These may be saved and restored by the kernel only when a thread is in the context save/restore mode. Access of these registers otherwise, will result in undeterministic behaviour.

Each thread has 8 registers. The granularity of access is always one full register, i.e., 256b. These registers must be accessed with a MOV with no predication, src modifiers or conditional modifiers. They

MUST be accessed in direct addressing more only. Access mode is ignored when reading/writing these registers.

These registers must be accessed in order, i.e., reads/writes are in order from msg0 to msg7.

On a context save, the message control register is written to the upper oword of a register.

On a context restore, the upper oword of register must be moved to message control register. However, since these move operations require data from lower oword of a register, it is required to move the upper oword to lower oword before moving to these registers.

Example

Context Save Cycle:

```
mov (8) r113.0:ud msg0.0:ud { NoMask }
```

The above operation saves content of fc0.0 to the upper oword of r113

Context Restore Cycle:

```
mov (4) rtmp.0:ud r113.4<4;4,1>:ud { NoMask }
mov (8) msg0.0:ud rtmp:ud { NoMask }
```

The above operation restores content from upper oword of r113 by moving to lower oword of rtmp.

Programming Note	
Context:	Message Control Registers
Message Control Registers must never be saved or restored	

SP Register

SP Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0110b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	2
Element Size:	64 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control	No
Indexable	No

The SP register can be accessed as a unsigned DWord integer. It is a read-write register, containing the current stack pointer, which is relative to the Generate State Base Address. The stack pointer is inserted into the message header when data is stored into scratch space as a stack. The stack pointer is managed by software. If the stack pointer exceeds the limit or the space allocated, an exception is triggered. See the Stack Pointer Exception in the Exceptions Section.

Register and Subregister Numbers for SP Register

RegNum[3:0]	SubRegNum[4:0]
0000b = sp All other encodings are reserved.	00000b = **sp:uq**. 01000b = **sp_limit:uq** All other encodings are reserved.

SP Register Fields

DWord	Bits	Description
0..1	63:48	**Reserved**. MBZ.
	47:0	**sp**. Specifies the current stack pointer. This pointer is relative to the General State Base Address. This register is initialized at thread load to the top of the per thread Scratch Space. The register is R/W. *sp = [scratch space pointer] + [scratch space] - 1* Alternatively, this register may be updated by Software to any flat address space. In such cases, the stack is NOT relative to the General State Base Address. Software must ensure that the address is exclusive for the thread.
2..3	63:48	**Reserved**. MBZ.
	47:0	**sp limit**. Specifies the upper limit for the stack pointer. This pointer is relative to the General State Base Address. This register is initialized at thread load to the limit allocated for stack in the state. See the GPGPU Thread Payload description for details. The register is R/W. *sp_limit = [scratch space pointer] + [stack space limit]* Alternatively, this register may be updated by software, similar to the sp register. In such cases, software is responsible for allocating the right thread stack pointer limit.

State Register

State Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	0111b
Number of Registers:	2
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	Byte
Write Mask Granularity:	N/A
SecHalf Control?	No
Indexable?	No

On a context restore, SIP must ensure all pending context restore cycles to the GRF are committed/completed before restoring SR1.

Example:

```
send (8)  rdest0 rsrc0 <msg_dscpt>
send (8)  rdest1 rsrc1 <msg_dscpt>
send (8)  rdest2 rsrc2 <msg_dscpt>
send (8)  rdest3 rsrc3 <msg_dscpt>
send (8)  rdest4 rsrc4 <msg_dscpt>
// the following mov's are required to ensure all previous reads are complete
mov (8) null rdest0
mov (8) null rdest1
mov (8) null rdest2
mov (8) null rdest3
mov (8) null rdest4
// restore the grfsb
mov (8) sr1 rsrc
```

Register and Subregister Numbers for State Register

RegNum[3:0]	SubRegNum[4:0]
0000b = sr0 All other encodings are reserved.	Valid encoding range: 00000b – 01100b All other encodings are reserved.
0001b = sr1 All other encodings are reserved.	Valid encoding range: 00000b All other encodings are reserved.

State Register Fields

DWord	Bits	Description
0 (sr0.0:ud)	31:28	**Reserved**. MBZ.
	27:24	**FFID** (Fixed Function Identifier). Specifies which fixed function unit generates the current thread. This field is set at thread dispatch and is forwarded on the message bus for all out-going messages from this thread.
	23	**Priority Class**. This field, when set, indicates the thread belongs to the high priority class, which has higher scheduling priority over any thread with this field cleared. The priority field below determines the relative priority within the same priority class. This field is initialized by the thread dispatcher at thread dispatch time and stays unchanged throughout the life span of the thread. This field is forwarded on the message bus to the message bus arbiter for all out-going messages. It serves as a priority hint for the target shared function. See the Shared Function chapters for whether and how a shared function uses this priority hint. 0 = Low priority class. 1 = High priority class.
	22:19	**Reserved**. MBZ.
	18:16	**Priority**. This field is the relative aging priority of the thread. This field indicates the 'age' of this thread relative to other threads within the EU. No two threads in the same EU can have the same priority number (independent of the priority class value). Within the same priority class, an older thread (with a larger priority number) has higher schedule priority over a younger thread. This field is set to zero at a thread's dispatch. During a thread's run time, this field may or may not be incremented when a new thread is dispatched to the same EU. It is only incremented when another thread's priority number is incremented and reaches the same value. For example, if currently there is a thread with priority 0 on an EU, then dispatching a new thread to that EU causes the old thread's priority number to increment to 1. However, if the active thread (assuming for simplicity that there is only one) on an EU has a priority number 1 (or 2 or 3), then dispatching a new thread to this EU does not change the old thread's priority number. As threads on an EU may terminate in arbitrary order, the exact number for a thread depends on the dynamic execution of threads. When thread context is saved and restored after pre-emption, the Priority is not restored to the original state. Instead the priority is initiated as if new threads were loaded.
	15:8	[15:14] Slice ID. [13:12] SubSlice ID.

DWord	Bits	Description
		[11:8] EUID.
	3	**Reserved**. MBZ.
	7:3	**Reserved**. MBZ.
	2:0	**TID** (The thread identifier). Specifies the thread slot that the current thread is assigned to. This field is set at thread dispatch.
1 (sr0.1:ud)	31:23	**FFTID** (Fixed Function Thread ID). There is no connection between this thread ID, assigned in fixed functions, and the TID assigned in the EUs.
	22	**Reserved.**
	21	**Hardware Defined State.** The byte is defined for hardware use only. The content is saved and restored in the event of mid thread preemption.
	20	**Page Fault Status.** This bit speficies if the thread has hit a page fault on a memory read, memory write or instruction fetch operation. The bit is cleared when the thread is restarted after a fault.
	19	**Reserved.**
	18:16	**Page Fault Code.** The fault code indicates the type of fault encountered.
	15:8	**Hardware Defined State.** The byte is defined for hardware use only. The content is saved and restored in the event of mid thread preemption.
	7:0	**IEEE Exception.** The exception bits are sticky bits set by the opcode when one of the exception is triggered. These bits are defined per thread and all channels update one sticky bit. These bits may be cleared by software or on a thread load. Updates to these bits may be turned OFF by the IEEE Exception trap enable in the CR register. When these bits are required as source of an operation, the previous instruction must use a {Switch}. This ensures all asynchronous flag updates are complete before using as source operand. The following table describes these bits:

Bits	Definition
[7:5]	Reserved
4	Inexact Exception
3	Overflow
2	Underflow
1	Divide by Zero
0	Invalid Operation

	20	**Page Fault Status.** This bit speficies if the thread has hit a page fault on a memory read, memory write or instruction fetch operation. The bit is cleared when the thread is restarted after a fault.
	19	**Reserved.**
	18:16	**Page Fault Code.** The fault code indicates the type of fault encountered.

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td>15:7</td><td>Hardware Defined State. The byte is defined for hardware use only. The content is saved and restored in the event of mid thread preemption.</td></tr>
<tr><td></td><td>6:0</td><td>IEEE Exception. The exception bits are sticky bits set by the opcode when one of the exception is triggered. These bits are defined per thread and all channels update one sticky bit. These bits may be cleared by software or on a thread load. Updates to these bits may be turned OFF by the IEEE Exception trap enable in the CR register. When these bits are required as source of an operation, the previous instruction must use a {Switch}. This ensures all asynchronous flag updates are complete before using as source operand. The following table describes these bits:
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>[6:5]</td><td>Reserved</td></tr>
<tr><td>4</td><td>Inexact Exception</td></tr>
<tr><td>3</td><td>Overflow</td></tr>
<tr><td>2</td><td>Underflow</td></tr>
<tr><td>1</td><td>Divide by Zero</td></tr>
<tr><td>0</td><td>Invalid Operation</td></tr>
</table></td></tr>
<tr><td rowspan="6">1
(sr0.1:ud)</td><td>31:22</td><td>FFTID (Fixed Function Thread ID). There is no connection between this thread ID, assigned in fixed functions, and the TID assigned in the EUs.</td></tr>
<tr><td>21</td><td>Reserved.</td></tr>
<tr><td>20</td><td>Page Fault Status. This bit speficies if the thread has hit a page fault on a memory read, memory write or instruction fetch operation.
The bit is cleared when the thread is restarted after a fault.</td></tr>
<tr><td>18:16</td><td>Page Fault Code. The fault code indicates the type of fault encountered.</td></tr>
<tr><td>15:7</td><td>Hardware Defined State. The byte is defined for hardware use only. The content is saved and restored in the event of mid thread preemption.</td></tr>
<tr><td>6:0</td><td>IEEE Exception. The exception bits are sticky bits set by the opcode when one of the exception is triggered. These bits are defined per thread and all channels update one sticky bit. These bits may be cleared by software or on a thread load. Updates to these bits may be turned OFF by the IEEE Exception trap enable in the CR register. When these bits are required as source of an operation, the previous instruction must use a {Switch}. This ensures all asynchronous flag updates are complete before using as source operand. The following table describes these bits:
<table>
<tr><th>Bits</th><th>Definition</th></tr>
<tr><td>[6:5]</td><td>Reserved</td></tr>
<tr><td>4</td><td>Inexact Exception</td></tr>
<tr><td>3</td><td>Overflow</td></tr>
<tr><td>2</td><td>Underflow</td></tr>
<tr><td>1</td><td>Divide by Zero</td></tr>
</table></td></tr>
</table>

DWord	Bits	Description
		0 — Invalid Operation
2 (sr0.2:ud)	31:0	**Dispatch Mask** (DMask). This 32-bit field specifies which channels are active at Dispatch time. This field is used by hardware to initialize the mask register. Format: U32
3 (sr0.3:ud)	31:0	**Vector Mask** (VMask). This 32-bit field contains, for each 4-bit group, the OR of the corresponding 4-bit group in the dispatch mask. This field is used by hardware to initialize the mask register. Format: U32
0 (sr1.0:ud)	31:0	**Hardware Defined State Register.** The contents of these register are hardware defined and are required only for handling page-fault. These bits are saved and restored by SIP when threads are pre-empted. Writes to these registers must follow the sequence described in 'send' instruction for the correct behavior of hardware.
1 (sr1.1:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0
2 (sr1.2:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0
3 (sr1.3:ud)	31:0	**Hardware Defined State Register.** Same as sr1.0

Implementation Restriction on Register Access: When the state register is used as a source and/or destination, hardware does not ensure execution pipeline coherency. Software must set the thread control field to *'switch'* for an instruction that uses state register as an explicit operand. This is important as the state register is an implicit source or destination for many instructions. For example, fields like IEEE Exception may be an implicit destination updated by multiple back to back instructions. Therefore, if the instructions updating the state register doesn't set *'switch'*, subsequent instructions may have undefined results.

Control Register

Control Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1000b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	4
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

The Control register is a read-write register. It contains four 32-bit subregisters that can be accessed individually.

Subregister *cr0.0:ud* contains normal operation control fields such as the floating-point mode and the accumulator disable. It also contains the master exception status/control field that allows software to switch back to the application thread from the System Routine.

Subregister *cr0.1:ud* contains the mask and status/control fields for all exceptions. The exception fields are arranged in significance-decreasing order from MSB to LSB. This arrangement allows the System Routine to use the *lzd* instruction to find the high priority exceptions and handle them first. As each exception is mapped to a single bit, another exception priority order may be implemented by software. The System Routine may choose to handle one exception at a time, by handling the exception detected by an *lzd*instruction and returning to the application thread. Or it may choose to handle all the concurrent exceptions, by looping through the exception fields until all outstanding exceptions are handled before returning back to the application thread.

Exception enable bits (bits 15:0 in *cr0.1:ud*) control whether an exception causes hardware to jump to the System Routine or not. Exception status and control bits (bits 31:16 in *cr0.1:ud*) indicate which exceptions have occurred, and are used by the system routine to clear the exception. Even if a given exception is disabled, the corresponding exception status and control bit still reflects its status, whether an exception event has occurred or not.

cr0.2:ud contains the **Application IP (AIP)** indicating the current thread IP when an exception occurs.

cr0.3:ud is reserved. Values written to this subregister are dropped; the result of reading from this subregister is unpredictable.

Fields in Control registers also reference a virtual register called **System IP (SIP)**. SIP is the virtual register holding the global System IP, which is the initial instruction pointer for the System Routine.

There is only one SIP for the whole system. It is virtual only from a thread's point of view, as it is not visible (i.e. not readable and not writeable) to the thread software executed on a GEN EU. It can only be accessed indirectly by the hardware to respond to exception events. Upon an exception, hardware performs some bookkeeping (e.g. saving the current IP into AIP) and then jumps to SIP. Upon finishing exception handling, the System Routine may return back to the application by clearing the Master Exception Status and Control field in *cr0*, which causes the hardware to load AIP to IP register. See the STATE_SIP command for how to set SIP.

The SIP is widened to 48 bits. However the EU still only uses the low 32 bits.

Register and Subregister Numbers for Control Register

RegNum[3:0]	SubRegNum[4:0]
0000b = cr0 All other encodings are reserved.	00000b = **cr0.0:ud**. It contains general thread control fields. 00100b = **cr0.1:ud**. It contains exception status and control. 01000b = **cr0.2:ud**. It contains AIP. All other encodings are reserved.

Control Register Fields

DWord	Bits	Description
0	31	**Master Exception State and Control.** This bit is the master state and control for all exceptions. Reading a 0 indicates that the thread is in normal operation state and a 1 means the thread is in exception handle state. Upon an exception event, hardware sets this bit to 1 and switches to SIP. Writing 1 to this bit has no effect. Writing 0 to this bit also has no effect if the previous value is 0. In both cases, the bit keeps the previous value. If the previous value of this bit is 1, software writing a 0 causes the thread to return to AIP. This transition is automatic – software does not have to move AIP to IP. The value of this bit then stays as 0. This bit is initialized to 0. 0 = The thread is in normal state. 1 = The thread is in exception state.
	30:16	**Reserved.** MBZ.
	14:13	**Reserved.** MBZ.
	12	**Reserved.** MBZ.
	11	**Reserved.** MBZ.
	10	**Reserved.** MBZ.
	9	**IEEE Exception Trap Enable.** This bit enables trapping IEEE exception flags. This control bit may updated by software. It is initially zero on thread load. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exceptions field of sr0.1. Note that IEEE floating-point exceptions do *not* transfer control to any handler.

DWord	Bits	Description
		0 = IEEE Exception flags are NOT trapped. 1 = IEEE Exception flags are trapped.
	7	**Single Precision Denorm Mode.** This bit determines how denormal numbers are handled for the F (Float) type when using the IEEE floating-point mode. It is ignored in the ALT floating-point mode, which always flushs denorms. This bit is initialized by Thread Dispatch. 0 = Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign. 1 = Allow denorm source values and denorm results.
	6	**Double Precision Denorm Mode.** This bit determines how denormal numbers are handled for the DF (Double Float) type. It is initialized by Thread Dispatch. 0 = Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign. 1 = Allow denorm source values and denorm results.
	5:4	**Rounding Mode.** This field specifies the FPU rounding mode. It is initialized by Thread Dispatch. 00b = Round to Nearest or Even (RTNE) 01b = Round Up, toward +inf (RU) 10b = Round Down, toward -inf (RD) 11b = Round Toward Zero (RTZ)
	3	**Vector Mask Enable (VME).** This bit indicates DMask or Vmask should be used by EU for execution. This bit is set by the Thread Dispatch. 0: Use Dispatch Mask (DMASK) 1: Use Vector Mask (VMASK)
	2	**Single Program Flow (SPF).** Specifies whether the thread has a single program flow (SIMDnxm with m = 1) or multiple program flows (SIMDnxm with m > 1). This bit affects the operation of all branch instructions. In Single Program Flow mode, all execution channels branch and/or loop identically. This bit is initialized by the Thread Dispatch. 0: Multiple Program Flows 1: Single Program Flow **Programming Restrictions:** Only H1/Q1/N1 are allowed in SPF mode.

DWord	Bits	Description
		Power Optimization: If an entire shader does not do SIMD branching, the driver can set the SPF bit to 1 to save power in HW. SPF mode must be set to 0 at the start of context restore.
	1	**Accumulator Disable.** This bit controls the update of the accumulator by the instruction field AccWrCtrl. If this bit is cleared, the accumulator is updated for all instructions with AccWrCtrl enabled. If set, the accumulator is disabled for all update operations, maintaining its value prior to being disabled. Setting this bit has no effect if the accumulator is the explicit destination operand for an instruction. This bit is initialized to 0. 0: Enable accumulator update. 1: Disable accumulator update. **Usage Notes:** This control bit is primarily designed for the System Routine. That routine is not expected to use the accumulator, though it may need to use instructions that implicitly update the accumulator. To use such instructions in the System Routine, but still preserve the accumulator contents on returning to the application kernel, the System Routine would either (a) save and restore the accumulator, or (b) prevent the accumulator from being unintentionally modified. This control bit has been added for the latter method. Software has the option to limit the setting of this control bit to strictly within the System Routine. If, by convention, this bit is clear within application kernels, the System Routine can simply set the bit upon entry and clear it before returning control to the application kernel. This usage model would not necessarily require cr0.0 to be saved/restored in the System Routine. However, if by convention application kernels are permitted to set this bit, then the System Routine is required to preserve the content of this bit.
	0	**Single Precision Floating Point Mode (FP Mode).** This bit specifies whether the current single-precision floating-point operation mode is IEEE mode (IEEE Standard 754) or the ALT (alternative mode). This bit does not affect the floating-point mode used for other floating-point data types. This bit is also forwarded on the message sideband for all out-going messages, for example, to control the floating-point mode of the Sampler. Software may modify this bit to dynamically switch between the two floating-point modes. This bit is initialized by Thread Dispatch. 0 = IEEE floating-point mode for the F (Float) type. 1 = ALT (alternative) floating-point mode for the F (Float) type.
	30	**External Halt Exception Status and Control**. This bit indicates the External Halt exception. It is set by EU hardware on receiving the broadcast External Halt signal. The System Routine should reset this bit before returning to an application

DWord	Bits	Description
		routine to avoid infinite loops. This bit may be set or cleared by software. This bit is initialized to 0.
	29	**Software Exception Control**. This bit is the control bit for software exceptions. Setting this bit to 1 in an application routine causes an exception. Clearing this bit in an application routine has no effect. Upon entering the system routine, the hardware maintains this bit as 1 to signify a software exception. The System Routine should reset this bit before returning to an application routine. This bit may be set or cleared by software. This bit is initialized to 0.
	28	**Illegal Opcode Exception Status**. This bit, when set, indicates an illegal opcode exception. The exception handler routine normally does not return back to the application thread upon an illegal opcode exception. Leaving this bit set has no effect on hardware; if system software adversely returns to an application routine leaving this bit set, it doesn't cause any exception. This bit should not be set by software or left set by the system routine to avoid confusion. This bit is initialized to 0.
	27	**Stack Overflow Exception Status**. This bit when set, indicates a stack overflow exception. The exception handler routine normally does not return back to the application thread upon a stack overflow exception. Leaving this bit set has no effect on hardware; if system software adversely returns to an application routine leaving this bit set, it doesn't cause any exception. This bit should not be set by software or left set by the system routine to avoid confusion. This bit is initialized to 0.
	25	**Context Save Status.** This bit when set, indicates a Context Save process has been initiated. The system routine must reset this bit after saving the context to terminate the thread.
	24	**Context Restore Status.** This bit when set, indicates a Context Restore process has been initiated. The system routine must reset this bit after restoring the context. The reset of this bit is required before invoking application routine.
	23:16	**Reserved**. MBZ.
	13	**Software Exception Enable**. This bit enables or disables the software exception. Enabling or disabling this bit may allow host software to turn on/off certain features (such as profiling) without changing the kernel program. This bit is initialized by the Thread Dispatcher. Format = ENABLED: 0: Disabled 1: Enabled

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td></td><td>12</td><td>Illegal Opcode Exception Enable. This bit specifies whether the illegal opcode exception is enabled or not. The Illegal opcode exception includes illegal opcodes and undefined opcodes, caused by bad programs or run-time data corruption.

This bit is initialized by the Thread Dispatcher.

Software should normally assign this bit in the interface descriptor. Even though this mechanism is provided to disable the illegal opcode exception, it should be used with extreme caution.

Format = ENABLED:

0: Disabled

1: Enabled</td></tr>
<tr><td></td><td>11</td><td>Stack Overflow Exception Enable. This bit specifies whether the stack overflow exception is enabled or not. The stack overflow exception includes an overflow or an underflow in the stack space allocated for the thread.

This bit is initialized by the Thread Dispatcher.

Software should normally assign this bit in the interface descriptor.

Format = ENABLED:

0: Disabled

1: Enabled</td></tr>
<tr><td></td><td>10:0</td><td>Reserved. MBZ.</td></tr>
<tr><td>2

(cr0.2:ud)</td><td>31:3</td><td>Application IP (AIP). This is the register storing the instruction pointer before an exception is handled. Upon an exception, hardware automatically saves the current IP into the AIP register, and then sets the Master Exception State and Control field to 1, which forces a switch to the System IP (SIP). The AIP register may contain either the pointer to the instruction that causes the exception (such as breakpoint) or the one after (such as masked stack overflow/underflow exceptions). This is shown in the following table, where IP is the instruction that generated the exception.

<table>
<tr><th>Exception Type</th><th>AIP Value</th></tr>
<tr><td>External Halt</td><td>N/A (1)</td></tr>
<tr><td>Software Exception</td><td>IP + 1</td></tr>
<tr><td>Illegal Opcode</td><td>IP</td></tr>
</table>
(1) External Halt exception is asynchronous and not associated with an instruction.

When the System Routine changes the Master Exception State and Control field from 1 to 0, hardware restores IP from this register. This field is writable allowing the returning IP to be altered after an exception is handled.</td></tr>
</table>

DWord	Bits	Description
	2:0	**Reserved.** MBZ.

Programming Note	
Context:	
Implementation Restriction on Register Access: When the control register is used as an explicit source and/or destination, hardware does not ensure execution pipeline coherency. Software must set the thread control field to *'switch'* for an instruction that uses control register as an explicit operand. This is important as the control register is an implicit source for most instructions. For example, fields like FPMode and Accumulator Disable control the arithmetic and/or logic instructions. Therefore, if the instruction updating the control register doesn't set *'switch'*, subsequent instructions may have undefined results.	

Notification Registers

Notification Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1001b
Number of Registers:	3
Default Value:	No
Normal Access:	RO (RW – Context save/restore only)
Elements:	3
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

There are three notification registers (*n0.0:ud*, *n0.1:ud, and n0.2:ud*) used by the *wait* instruction. These registers are read-only, except under context restore, and can be accessed in 32-bit granularity. Write access to this register is allowed only when context is restored.

It should be noted that in the extreme case, it is possible to have more notifications to a thread than the maximum allowed number of concurrent threads in the system. Therefore, the range of the thread-to-thread notification count in n0, is larger than the maximum number of threads computed by EUID * TID.

There is only one bit for the host-to-thread notification count in n1.

When directly accessed, this register is read-only. If the value is non zero, the only way to alter the value is to use the wait instruction to decrement the value until zero is reached. A wait instruction on a zero notification subregister causes the thread to stall, waiting for a notification signal from outside targeting the same subregister. See the wait instruction for details.

Implementation Restriction: The notification registers are initialized to 0 after hardware/software reset. However, these registers are not reset at thread dispatch time.

Register and Subregister Numbers for Notification Registers

RegNum[3:0]	SubRegNum[4:0]
0000b = n0 All other encodings are reserved.	00000b = n0.0:ud 00100b = n0.1:ud 01000b = n0.2:ud All other encodings are reserved.

Notification Register 0 Fields

DWord	Bits	Description
0	31:16	**Reserved**. MBZ.
	15:0	**Thread to Thread Notification Count**. This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See the WAIT instruction for details. Format: U16

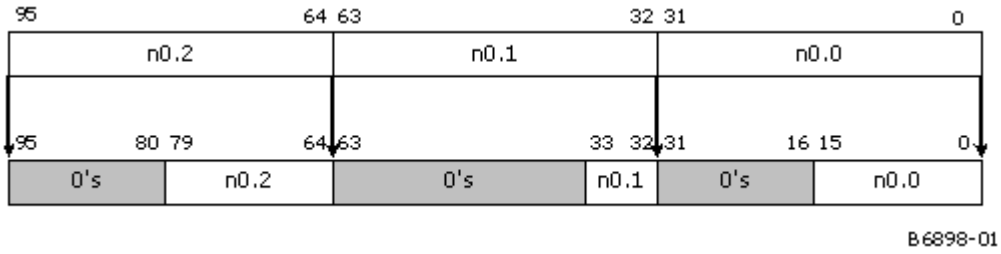
Notification Register 1 Fields

DWord	Bits	Description
0	31:1	**Reserved**. MBZ.
	0	**Host to Thread Notification**. This register is used by the WAIT instruction for host-to-thread synchronization via MMIO registers. Format: U1

Notification Register 2 Fields

DWord	Bits	Description
0	31:16	**Reserved**. MBZ.
	15:0	**Thread to Thread Notification Count**. This register is used by the WAIT instruction for thread-to-thread synchronization. The value read from this register specifies the outstanding notifications received from other threads. It can be changed indirectly by using the WAIT instruction. See the WAIT instruction for details. Format: U16

Format of the Notification Register



IP Register

IP Register Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1010b
Number of Registers:	1
Default Value:	Provided by the Dispatcher
Normal Access:	RW
Elements:	1
Element Size:	32 bits
Element Type:	UD
Access Granularity:	DWord
Write Mask Granularity:	DWord
SecHalf Control?	No
Indexable?	No

The ip register can be accessed as a 32-bit quantity. It is a read-write register, containing the current instruction pointer, which is relative to the **Generate State Base Address**. Reading this register returns the instruction pointer of the current instruction. The 3 LSBs are always read as zero. Writing this register causes program flow to jump to the new address. When it is written, the 3 LSBs are dropped by hardware.

Register and Subregister Numbers for IP Register

RegNum[3:0]	SubRegNum[4:0]
0000b = ip All other encodings are reserved.	00000b = **ip:ud** All other encodings are reserved.

IP Register Fields

DWord	Bits	Subfield Description
0	31:3	**Ip**. Specifies the current instruction pointer. This pointer is relative to the **General State Base Address**.
	2:0	**Reserved**. MBZ.

TDR Registers

TDR Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1011b
Number of Registers:	8
Default Value:	No
Normal Access:	RO/CW
Elements:	8
Element Size:	16 bits
Element Type:	UW
Access Granularity:	Word
Write Mask Granularity:	Word
SecHalf Control?	No
Indexable?	No

There are 8 thread dependency registers (tdr0.0:uw to tdr0.7:uw) used by HW for the *sendc* instruction. These registers are read-only and can be accessed in 16-bit granularity.

The thread dependency registers are read only, the valids can only be set with a thread dispatch, and are reset by broadcasting end of thread messages after a thread retired. The FFTID's can only be changed with a thread dispatch. Any write into any of the TDR registers will clear the valid bit for the particular TDR if the write enable is true, the FFTID portion is strictly read only.

Register and Subregister Numbers for TDR Registers

RegNum[3:0]	SubRegNum[4:0]
1011b = tdr0 All other encodings are reserved.	00000b = **tdr0.0:uw** 00010b = **tdr0.1:uw** 00100b = **tdr0.2:uw** 00110b = **tdr0.3:uw** 01000b = **tdr0.4:uw** 01010b = **tdr0.5:uw** 01100b = **tdr0.6:uw** 01110b = **tdr0.7:uw** All other encodings are reserved.

TDR Registers Fields [CHV, BSW]

DWord	Bits	Description
3	31	**Valid7**. This field indicates whether the thread specified by FFTID7 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID7**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid6**. This field indicates whether the thread specified by FFTID6 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID6**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
2	31	**Valid5**. This field indicates whether the thread specified by FFTID5 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID5**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid4**. This field indicates whether the thread specified by FFTID4 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID4**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
1	31	**Valid3**. This field indicates whether the thread specified by FFTID3 is still in-flight.
	30:25	**Reserved**. MBZ
	24:16	**FFTID3**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid2**. This field indicates whether the thread specified by FFTID2 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID2**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
0	31	**Valid1**. This field indicates whether the thread specified by FFTID1 is still in-flight.

	30:25	**Reserved**. MBZ
	24:16	**FFTID1**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9
	15	**Valid0**. This field indicates whether the thread specified by FFTID0 is still in-flight.
	14:9	**Reserved**. MBZ
	8:0	**FFTID0**. This field is the FFTID of the third thread that the current thread depends on. It can be changed by the end of thread broadcasting messages. Format: U9

Performance Registers

Performance Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1100b
Number of Registers:	1
Default Value:	0h
Normal Access:	RO/RW
Elements:	3
Element Size:	32 bits
Element Type:	UD

Timestamp Register

This register is a low latency timestamp source, "TM", available as part of a thread's Architectural Register File (ARF). This is a is free running counter, 64b in size, and exposed to the ISA as individual 32b high 'TmHigh' and low 'TmLow' unsigned integer source operands. As part of the EU's register space, access to the timestamp has a low and deterministic latency and therefore can be used for intra-kernel high resolution performance profiling.

The TM features provides a 1-bit indicator 'TmEvent' which identifies the occurrence of a time-impacting event such as context switch or frequency change since the last time any portion of the Timestamp register value was read by that thread. Software that uses the Timestamp capability should check this bit to identify when a relative time calculation may be suspect. To properly use this additional information, the instrumentation code should operate on the Timestamp register value as a whole (i.e. as an 8 dword register) so that the 64b time and this 1b value are captured simultaneously, as opposed to 32b portions, to eliminate the chance of missing a TmEvent that might occur between accesses to 32b portions of this register.

Programming Note	
Context:	Performance Registers
The Timestamp register is saved as part of thread state on context-save, but only 'TmEvent' is restored (and technically always restored to '1' as a context switch had occurred).	

Register and Subregister Numbers for Performance Register

RegNum[3:0]	SubRegNum[4:0]
0000b = *tm0* All other encodings are reserved.	00000b = **tm0.0:ud.** 00100b = **tm0.1:ud.** 01000b = **tm0.2:ud** 01100b = **tm0.3:ud** 10000b = **tm0.4:ud** **All other encodings are reserved.**

Performance Register Fields

DWord	Bits	Description
0 (tm0.0:ud)	31:0	**TmLow.** The lower 32b of the 64b timestamp value sourced from Cr clock. Read-only. Format: U32
1 tm0.1:ud	31:0	**TmHigh.** The upper 32b of the 64b timestamp value sourced from Cr clock. Read-only. Format: U32
2 tm0.2:ud	31:1	**Reserved**
	0	**TmEvent.** Indicates a discontinuous time-impacting event (e.g. context switch, frequency change) occurred since any portion of the Timestamp register was last read, thus making any relative duration calculation based on this counter suspect. This bit is reset at the time a new thread is loaded, and on each read of any portion of the 'Timestamp' register.
3 tm0.3 (pm0)	31:0	Undefined Format: U32

Flow Control Registers

Flow Control Registers Summary

Attribute	Value
ARF Register Type Encoding (RegNum[7:4]):	1101b
Number of Registers:	39
Default Value:	None
Normal Access:	RW*

Register and Subregister Numbers for Flow Conrol Registers

RegNum[3:0]	SubRegNum[4:0]
0000b = fc0	00000b-11111b = **fc0.0–fc0.31**.
0001b = fc1	00000b = **fc1.0**. All other encodings are reserved.
0010b = fc2	00000b = **fc2.0**. All other encodings are reserved.
0011b = fc3	00000b = **fc3.0**. 00001b = **fc3.1**. 00010b = **fc3.2**. 00011b = **fc3.3**. All other encodings are reserved.
0100b = fc4	00000b = **fc4.0**. All other encodings are reserved.

These are special hardware registers used in handling flow control operations. These registers may be accessed ONLY in context save/restore operation using the SIP. These registers are accessed with the 'MOV' opcode. Use of any other opcode or access of these registers in non-context save/restore modes may result in undeterministic behaviour of hardware.

These registers are accessed as 256b registers. Parts of the 256b register may be redundant, depending on the hardware implementation of each register. The fields "RegNum" and "SubRegNum" are used together to address these registers.

When restoring to fc4.0, the DW0 of the register must be copied to DW1 of the register before restoring to fc4.0.

Example:

```
mov (8) fc4.0 r4.0
```

This must be replaced with:

```
mov (1) r4.1:ud r4.0:ud
mov (8) fc4.0 r4.0
```

Once context save of fc0 is done, 0 must be copied to fc0 register.

On a context save, the flow control register is written to the upper oword of a register.

On a context restore, the upper oword of register must be moved to flow control register. However, since these move operations require data from lower oword of a register, it is required to move the upper oword to lower oword before moving to these registers.

Example:

Context Save Cycle:

```
mov    (8)  r113.0:ud       fc0.0:ud        { NoMask }
```

The above operation saves content of fc0.0 to the upper oword of r113.

Context Restore Cycle:

```
mov    (4)  rtmp.0:ud r113.4<4;4,1>:ud      { NoMask }
mov    (8)  fc0.0:ud rtmp:ud                          { NoMask }
```

The above operation restores content from upper oword of r113 by moving to lower oword of rtmp.

On a context restore, the sequence of operations shown below MUST be used to restore Flow Control Register. The registers used are as an example.

```
// Restore fc0.28 to fc0.31 into registers r100 to r103 (Send operation)
mov    (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov    (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov    (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov    (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov    (2)  r104.1:ud r100.0<2;2,1>:ud { NoMask }
mov    (2)  r105.1:ud r101.0<2;2,1>:ud { NoMask }
mov    (2)  r106.1:ud r102.0<2;2,1>:ud { NoMask }
mov    (2)  r107.1:ud r103.0<2;2,1>:ud { NoMask }

mov   (8) fc0.28:ud r104:ud         { NoMask }
mov   (8) fc0.29:ud r105:ud         { NoMask }
mov   (8) fc0.30:ud r106:ud         { NoMask }
mov   (8) fc0.31:ud r107:ud         { NoMask }

// Restore fc0.24 to fc0.27 into registers r100 to r103 (Send operation)
mov    (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov    (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov    (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov    (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov    (2)  r104.2:ud r100.0<2;2,1>:ud { NoMask }
mov    (2)  r105.2:ud r101.0<2;2,1>:ud { NoMask }
mov    (2)  r106.2:ud r102.0<2;2,1>:ud { NoMask }
mov    (2)  r107.2:ud r103.0<2;2,1>:ud { NoMask }

mov   (8) fc0.24:ud r104:ud         { NoMask }
mov   (8) fc0.25:ud r105:ud         { NoMask }
mov   (8) fc0.26:ud r106:ud         { NoMask }
mov   (8) fc0.27:ud r107:ud         { NoMask }

// Restore fc0.20 to fc0.23 into registers r100 to r103 (Send operation)
mov    (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov    (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov    (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov    (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }
```

```
mov     (2)  r104.3:ud r100.0<2;2,1>:ud { NoMask }
mov     (2)  r105.3:ud r101.0<2;2,1>:ud { NoMask }
mov     (2)  r106.3:ud r102.0<2;2,1>:ud { NoMask }
mov     (2)  r107.3:ud r103.0<2;2,1>:ud { NoMask }

mov    (8) fc0.20:ud r104:ud         { NoMask }
mov    (8) fc0.21:ud r105:ud         { NoMask }
mov    (8) fc0.22:ud r106:ud         { NoMask }
mov    (8) fc0.23:ud r107:ud         { NoMask }

// Restore fc0.16 to fc0.19 into registers r100 to r103 (Send operation)
mov     (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov     (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov     (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov     (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov     (2)  r104.4:ud r100.0<2;2,1>:ud { NoMask }
mov     (2)  r105.4:ud r101.0<2;2,1>:ud { NoMask }
mov     (2)  r106.4:ud r102.0<2;2,1>:ud { NoMask }
mov     (2)  r107.4:ud r103.0<2;2,1>:ud { NoMask }

mov    (8) fc0.16:ud r104:ud         { NoMask }
mov    (8) fc0.17:ud r105:ud         { NoMask }
mov    (8) fc0.18:ud r106:ud         { NoMask }
mov    (8) fc0.19:ud r107:ud         { NoMask }

// Restore fc0.12 to fc0.15 into registers r100 to r103 (Send operation)
mov     (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov     (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov     (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov     (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov     (2)  r104.5:ud r100.0<2;2,1>:ud { NoMask }
mov     (2)  r105.5:ud r101.0<2;2,1>:ud { NoMask }
mov     (2)  r106.5:ud r102.0<2;2,1>:ud { NoMask }
mov     (2)  r107.5:ud r103.0<2;2,1>:ud { NoMask }

mov    (8) fc0.12:ud r104:ud         { NoMask }
mov    (8) fc0.13:ud r105:ud         { NoMask }
mov    (8) fc0.14:ud r106:ud         { NoMask }
mov    (8) fc0.15:ud r107:ud         { NoMask }

// Restore fc0.8 to fc0.11 into registers r100 to r103 (Send operation)
mov     (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov     (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov     (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov     (4)  r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov     (2)  r104.6:ud r100.0<2;2,1>:ud { NoMask }
mov     (2)  r105.6:ud r101.0<2;2,1>:ud { NoMask }
mov     (2)  r106.6:ud r102.0<2;2,1>:ud { NoMask }
mov     (2)  r107.6:ud r103.0<2;2,1>:ud { NoMask }

mov    (8) fc0.8:ud r104:ud         { NoMask }
mov    (8) fc0.9:ud r105:ud         { NoMask }
mov    (8) fc0.10:ud r106:ud        { NoMask }
mov    (8) fc0.11:ud r107:ud        { NoMask }

// Restore fc0.4 to fc0.7 into registers r100 to r103 (Send operation)
mov     (4)  r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov     (4)  r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov     (4)  r102.0:ud r102.4<4;4,1>:ud      { NoMask }
```

```
mov     (4)   r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov     (1)   r104.7:ud r100.0:ud { NoMask }
mov     (1)   r104.0:ud r100.1:ud  { NoMask }
mov     (1)   r105.7:ud r101.0:ud { NoMask }
mov     (1)   r105.0:ud r101.1:ud  { NoMask }

mov     (1)   r106.7:ud r102.0:ud { NoMask }
mov     (1)   r106.0:ud r102.1:ud  { NoMask }
mov     (1)   r107.7:ud r103.0:ud { NoMask }
mov     (1)   r107.0:ud r103.1:ud  { NoMask }

mov    (8)  fc0.4:ud r104:ud         { NoMask }
mov    (8)  fc0.5:ud r105:ud         { NoMask }
mov    (8)  fc0.6:ud r106:ud         { NoMask }
mov    (8)  fc0.7:ud r107:ud         { NoMask }

// Restore fc0.0 to fc0.3 into registers r100 to r103 (Send operation)
mov     (4)   r100.0:ud r100.4<4;4,1>:ud      { NoMask }
mov     (4)   r101.0:ud r101.4<4;4,1>:ud      { NoMask }
mov     (4)   r102.0:ud r102.4<4;4,1>:ud      { NoMask }
mov     (4)   r103.0:ud r103.4<4;4,1>:ud      { NoMask }

mov    (8)  fc0.0:ud r100:ud         { NoMask }
mov    (8)  fc0.1:ud r101:ud         { NoMask }
mov    (8)  fc0.2:ud r102:ud         { NoMask }
mov    (8)  fc0.3:ud r103:ud         { NoMask }
```

Immediate

Two forms of immediate are provided as a source operand: scalar and vector.

Project	Description
CHV, BSW	The immediate field may be 64 bits or 32 bits. For a word, unsigned word, or half-float immediate data, software must replicate the same 16-bit immediate value to both the lower word and the high word of the 32-bit immediate field in a GEN instruction. The 64-bit immediate takes up two DWords of the instruction bit field. Hence a 64-bit immediate is supported ONLY for a MOV operation. The field is denoted by ***imm32:type*** for 32-bit immediates and ***imm64:type*** for 64-bit immediates.

Project	Description
CHV, BSW	For a scalar immediate, the numeric data types supported are ***:uw, :w, :ud, :d, :uq, :q*** for integers AND ***:hf, :f, :df*** for floats. Refer to the Instruction Machine format topics for the encoding of these immediates.

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. Both integer and float immediate vectors are supported.

An immediate integer vector is denoted by type **v** or **uv** as *imm32:v* or *imm32:uv*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Refer to the Numeric DataType topic for description of the packing of vector integers to a DWord.

An immediate float vector is denoted by type **vf** as *imm32:vf*, where the 32-bit immediate field is partitioned into 4 8-bit subfields. Refer to the Numeric DataType topic for the description of the packing of vector floats to a DWord.

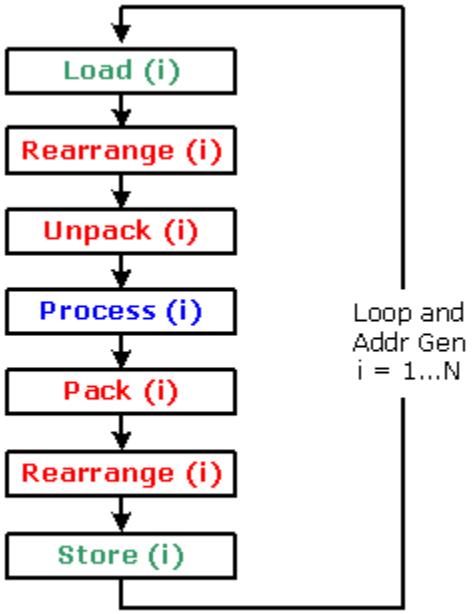
When an immediate vector is used in an instruction, the destination must be 128-bit aligned with destination horizontal stride equivalent to a word for an immediate integer vector (***v***) and equivalent to a DWord for an immediate float vector (***vf***).

Region Parameters

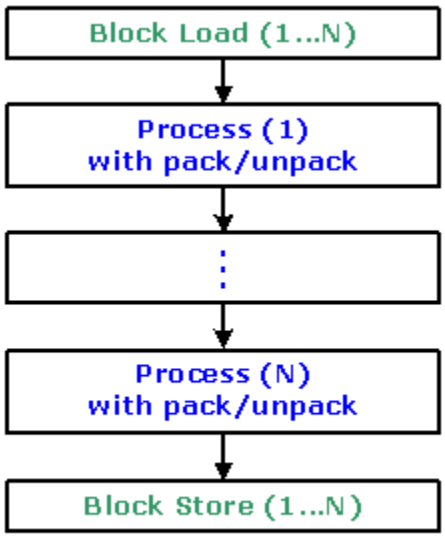
Unlike conventional SIMD architectures where an N-bit wide SIMD instruction can only operate on N-bit aligned SIMD data registers, a region-based register addressing scheme is employed in GEN architecture. The region-based register addressing capability significantly improves the SIMD computation efficiency by providing per-instruction-based multiple data gathering from register file. This avoids instruction overhead to perform data pack, unpack, and shuffling, which has been observed on other SIMD architectures. One benefit of such capability is allowing various kinds of 3D Graphics API Shader compute models to run efficiently on GEN. Another benefit is allowing high throughput of media applications, which tend to operate on byte or word data elements.

This can be illustrated by the example shown in *Region Parameters* and *Region Parameters*. As shown in *Region Parameters*, a sequence of SIMD instruction is executed on a conventional load/store based superscalar machine with SIMD instruction extension. The data parallelism can be achieved by first level of loop unrolling. As shown, there is a second level of loop for the task. Before a given SIMD compute instruction, *Process (i)*, can proceed, there might be a load, a data rearrange and a data unpack (and conversion) instruction to load and prepare the input data. After the compute instruction is complete, it might also require pack, re-arrange and store instructions, to format and save the same to memory. At the loop, other scalar computations such as loop count and address generation may be needed. For the same program, when the data can fit in the large GEN GRF register file, the outer loop may be unrolled for GEN. Here one or a few block loads (using *send* instruction) may be sufficient to move the working set into GRF. Then the data shuffle can be combined with the processing operation with region-based addressing capability. Per operand float type and mixed data type operation may also allow GEN to combine data conditioning operations with computing operations. These techniques in GEN architecture help to achieve high compute efficiency and throughput for graphics and media applications.

Conventional SIMD Instruction Sequence



GEN SIMD Instruction Sequence for the Same Program



B6900-01

In a GEN instruction, each operand defines a region in the register file. A region may contain multiple data elements. Each data element is assigned to an execution channel in the EU. The total number of data elements of a region is called the **size** of the region, or the size of the operand. The number of execution channels is called the **execution size** (*ExecSize*), which is specified in the instruction word. ExecSize determines the size of region for source and destination operands in an instruction.

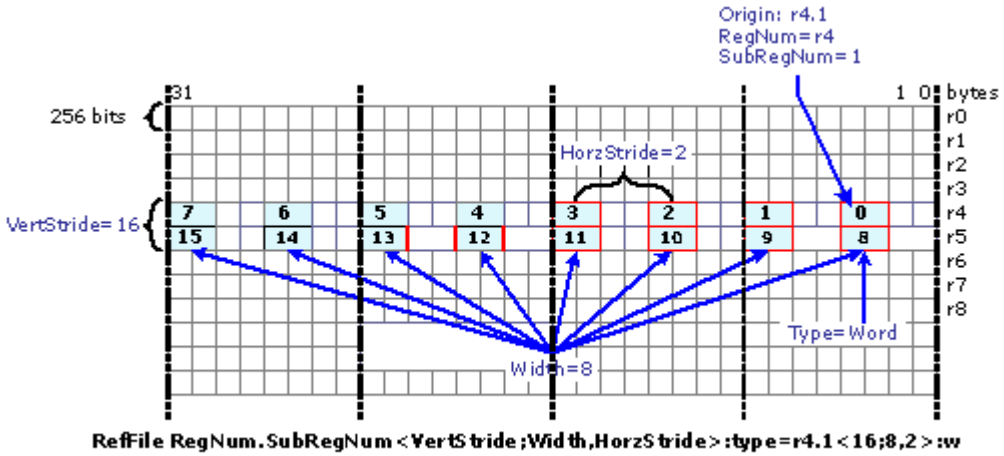
- For an instruction with two source operands, the sizes of the two source operands must be the same.
- The size of a destination operand generally matches the execution size, therefore equals to the number of source operand(s) in the same instruction.
    - Exception of this rule is present for the integer reduction instructions (such as sad2 and sada2) where the destination area is smaller than the source area.

Regions are **generalized 2-dimensional** (2D) arrays in row-major order. The first dimension is named the **horizontal** dimension (data elements within a row) and the second dimension is termed the **vertical** dimension (data elements in a column). Here, horizontal/vertical and row/column are just symbolic notations.

Description
When the GRF registers are viewed as a row-major 2D array of memory, such a notation normally matches well with the geometric locations of the data elements of an operand.

However, as the register region is fully described by the parameters discussed below, the data elements of a register region may not form a regular rectangular shape. For example, Vertical Stride parameter is allowed to be smaller than Horizontal Stride, making the rows of a register region interleave with each other. It should also note that the meanings of horizontal/vertical here is different than that used for the flag control in Section Flag Register

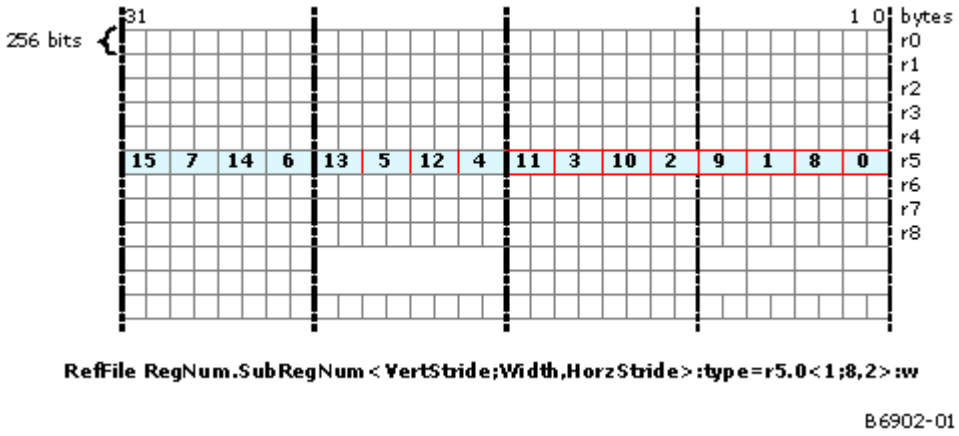
An example of a register region (*r4.1<16;8,2>:w*) with 16 elements



Region Parameters shows another example where the rows are interleaved. The region, having word data elements, starts at location r5.0:w. HorzStride, the distance within a row, is 2 words. So the second element (channel number 1) is at location 5.2:w. And there are 8 elements per row. VertStride, the distance between two rows, is only 1 word, which is less than HorzStride. Therefore, the first element of the second row (channel number 8) is at r5.1:w, just next to channel number 0. It is clear from the picture that the two rows are interleaved.

By varying the region parameters, reader may construct other configurations. The next section provides more details on the region-based register addressing. However, there are restrictions imposed by hardware implementation, which can be found in the later sections of this chapter.

A 16-element register region with interleaved rows (*r5.0<1;8,2>:w*)



Without considering the source channel swizzle and destination register region description, the above row-major-order region description provides the data assignment to each execution channel. The

following pseudo code computes the addresses of data elements assigned to execution channels for a special case when the destination register is aligned to 256-bit register boundary.

```
// Input: Type: ub | b | uw | w | ud | d | f | v
//RegNum: In unit of 256-bit register
//SubRegNum: In unit of data element size
//ExecSize, Width, VertStride, HorzStride: In unit of data elements
// Output: Address[0:ExecSize-1] for execution channels
int ElementSize = (Type=="b"||Type=="ub") ? 1 : (Type=="w"|Type=="uw") ? 2 : 4;
int Height = ExecSize / Width;
int Channel = 0;
int RowBase = RegNum«5 + SubRegNum * ElementSize;
for (int y=0; y<Height; y++) {
        int Offset = RowBase;
        for (int x=0; x<Width; x++) {
Address [Channel++] = Offset;
Offset += HorzStride*ElementSize;
        }
        RowBase += VertStride * ElementSize;
}
```

As *HorzStride* and *VertStride* are specified independently (note that *VertStride* might be smaller than or equal to *HorzStride*), the region may take various shapes from a replicated scalar, a replicated vector, a vector of replicated scalars, a sliding window, to a non-overlapped 2D array.

A region-based description of a destination operand can take the following simplified format

RegFile RegNum.SubRegNum<HorzStride>:type

The destination operand is only allowed to have a 1 dimensional region. The Register Region Origin and Type are the same as for a source operand. The total number of elements is given by *ExecSize*. However, only *HorzStride* is required to describe the 1D region, not *VertStride* and *Width*.

As a source register region may cross multiple physical GRF registers, an instruction with such source operands may take more than two execution cycles to gather source data elements for execution. The destination register region is restricted to be within a physical GRF register. In other words, destination scatter writes over multiple registers are not supported.

Region Addressing Modes

There are two different register addressing modes: Direct register addressing and register-indirect register addressing. Depending on the register region description, the register-indirect register addressing mode can be further divided into three usages: 1x1 index region where only the origin of register region is provided by the address register, Vx1 index region where the offset of each row of the register region is provided by an address register, VxH index region where the offset of each data element is provided by an address register.

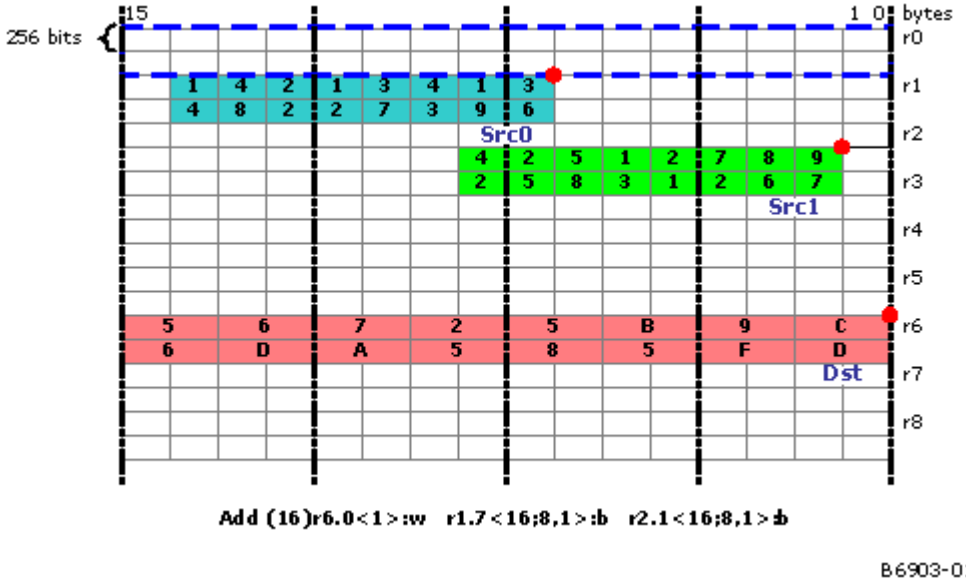
Direct Register Addressing

In this mode, all register region parameters are specified for an operand using fields in the instruction word.

Direct Register Addressing and *Direct Register Addressing* are two examples of direct register addressing.

For the example in *Direct Register Addressing*, all operands are 2D rectangular regions having the same size of 16 data elements. The two source operands, *Src0* and *Src1*, have 16 bytes. The destination operand, *Dst*, has 16 words. There are 8 elements in a row for *Src0* and *Src1*. The vertical stride of 16 bytes for *Src0* and *Src1* indicates that the first element and the 9'th element are 16 bytes apart in the register file. Note that *Src0* falls into the 256-bit physical GRF register starting at r1.0, but Src1 crosses the 256-bit physical GRF register boundary between r2 and r3. The numbers in the shaded regions are the values of the data elements. Observing the upper right corners of the source/destination regions (first data element), we have C = 3+9.

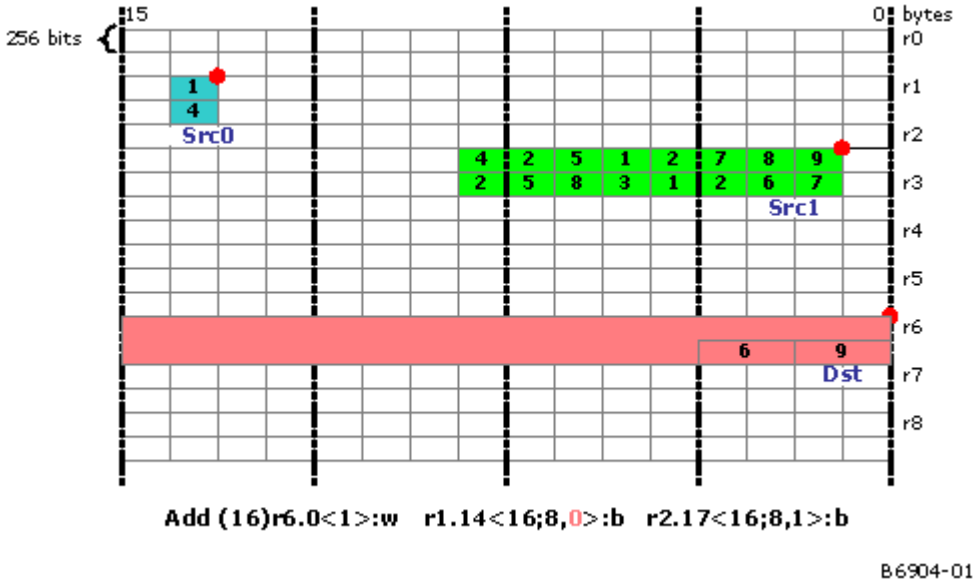
A region description example in direct register addressing



For the example in *Direct Register Addressing*, the sizes of areas of *Src0* and *Src1* are the same, but *Src0* contains a vector of replicated scalars. With HorzStride = 0 and Width = 8, the first row of 8 elements in Src0 is a replication of the byte at r1.14. Comparing *ExecSize* of 16 to Width of 8 indicates that there is a

second row of 8 elements in *Src0*. With VertStride = 16, the second row in *Src0* is a replication of the byte at r1.20 (20 = 14+16). Effectively, the 16 data elements of *Src0* are {1,1,1,1,1,1,1,1, 4,4,4,4,4,4,4,4}.

A region description example in direct register addressing with src0 as a vector of replicated scalars

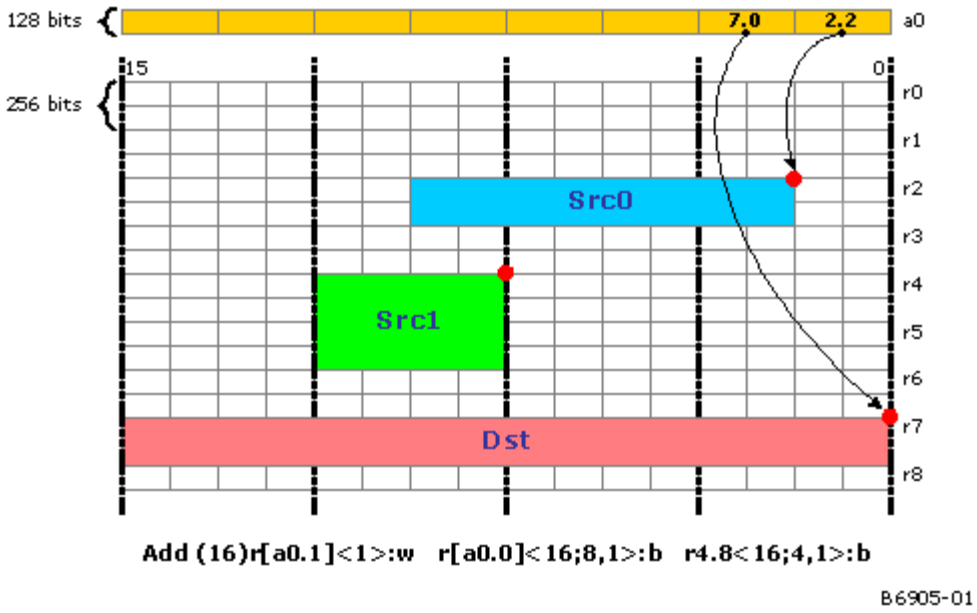


Register-Indirect Register Addressing with a 1x1 Index Region

In the register-indirect register addressing mode with 1x1 index region, the region origin is provided by the content of the address register, the rest of region parameters are provided by the fields in the instruction word.

Register-Indirect Register Addressing with a 1x1 Index Region depicts an example for this addressing mode. For example, the presence of a full region description <16;8,1> for Src0 indicates that only the origin of the region is provided by the address register a0.0.

An example illustrating register-indirect register addressing mode with a 1x1 index region



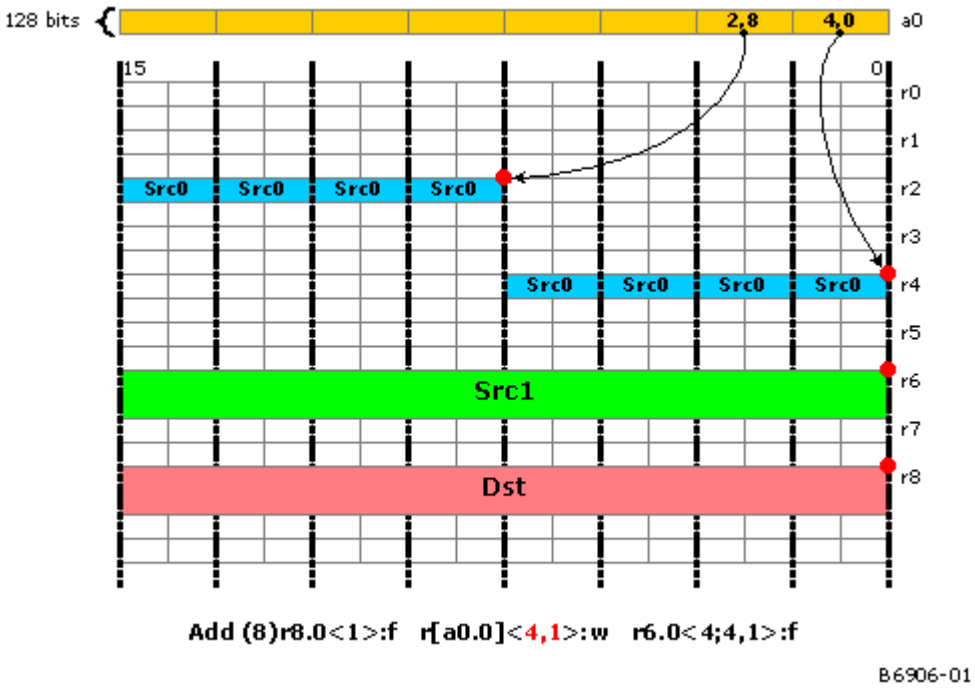
Register-Indirect Register Addressing with a Vx1 Index Region

In the register-indirect register addressing mode with Vx1 index region, the horizontal dimension is described by the fields in the instruction word and the vertical dimension is described by an address register region. Specifically, the origin of each row of the data region is provided by the contents of an address register region. The rows are described by the width and the horizontal stride. The first address register is provided and the following contiguous address registers are for the following rows. The total number of address registers used is inferred from the parameters *ExecSize* and *Width*.

Within the 16-bit address register, bits 15:5 determine RegNum and bits 4:0 determine SubRegNum.

An example is provided in *Register-Indirect Register Addressing with a Vx1 Index Region*. The assembly syntax notion of a register region without vertical stride, <4,1>, corresponding to the special encoding of vertical stride of 0xF in the instruction word, indicates the VxH or Vx1 mode of indirect register addressing. In this case, the origin for each row of src0 is provided by the address register. As ExecSize/Width = 2, there are two address registers a0.0 and a0.1, each pointing to a row of 4 data elements.

An example illustrating register-indirect-register addressing mode with a Vx1 index region (src0)

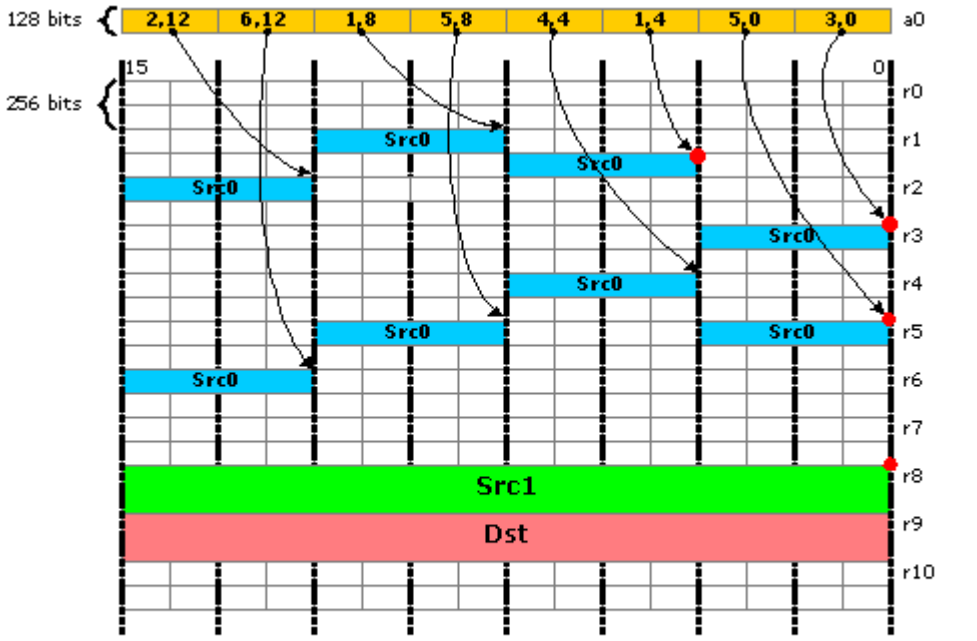


Register-Indirect Register Addressing with a VxH Index Region

In the register-indirect register addressing mode with VxH index region, the position of each data element is provided by the contexts in an address register region. This mode has the identical syntax as the Vx1 index region mode, and in fact, can be viewed as a special case of the Vx1 mode. When *Width* of the region is 1, the number of address registers used equals *ExecSize*.

An example is provided in *Register-Indirect Register Addressing with a VxH Index Region*. The absent of vertical stride in the region description <1,0> with width = 1 indicates that the origin for each row of 1 data element of Src0 is provided by the address register. As ExecSize/Width = 8, there are 8 address registers from a0.0 to a0.7, each pointing to a single data elements.

An example illustrating register-indirect register addressing mode with a VxH index region (Src0).



Add (8)r9.0<1>:f r[a0.0]<1,0>:f r8.0<4;4,1>:f

B6907-01

Access Modes

There are two basic GEN register access modes controlled by a single bit instruction subfield called Access Mode.

- 16-byte Aligned Access Mode (**align16**): In this mode, the origins of all operands (sources and destination), whether it is by direct addressing or register-indirect addressing, are 16-byte aligned. For example a row in the region description starts at 16-bype aligned and the width the row must be 4 and the 4 data elements within a row must span 16-bytes. In this access mode (and with other restrictions put forward later), full-channel swizzle for both source operands and per-channel mask for destination operand are supported on a 4-component basis. In other words, the control and setting of full source swizzle and destination mask are repeated for every 4 components up to total of *ExecSize* channels.
  - The **align16** access mode can be used for AOS operations. See examples provided in the Primary Usage Model section for SIMD4x2 and SIMD4x1 modes of operation to support 3D API Vertex Shader and Geometric Shader execution.
- 1-byte Aligned Access Mode (**align1**): In this mode, the origins of all operands may be aligned to their data type and could be 1-byte if the operand is of byte type. In this access mode, full region register descriptions are supported, however, source swizzle or destination mask are not supported.
  - The **align1** access mode can be used for SOA operations. See examples provided in the Primary Usage Model section for SIMD8 and SIMD16 modes of operation to support 3D API Pixel Shader. Many media applications also operate well in **align1** access mode.

Execution Data Type

The GEN architecture carries out arithmetic and logical operations using a smaller set of data types than the variety supported as source or destination operands. These are the *execution data types*. A particular arithmetic or logical instruction has one execution data type, from those listed in the table.

Execution Data Types

Type	Description
W	Word. 16-bit signed integer.
D	Doubleword. 32-bit signed integer.
Q	Quadword. 64-bit signed integer.
F	Float. 32-bit single precision floating-point number.
DF	Double Float. 64-bit double precision floating-point number.
HF	Half Float. 16-bit half precision floating-point number.

The following rules explain the conversion of multiple source operand types, possibly a mix of different types, to one common execution type:

- For floating-point sources, all source operands must have the same floating-point type, with the exceptions below:
  - A two-source floating-point instruction can have Float as the src0 type and VF (Packed Restricted Float Vector) as the immediate src1 type.

    When single precision and half precision floats are mixed between source operands or between source and destination operand. In such cases, singple precision float is the execution datatype.
- Mixing floating-point and integer source types is not allowed. Either all source types must be one floating-point type or all source types must be integer types.
- Unsigned integers are converted to signed integers.
- Byte (B) or Unsigned Byte (UB) values are converted to a Word or wider integer execution type.
- If source operands have different integer widths, use the widest width specified to choose the signed integer execution type.

Note that when the execution data type is an integer type, it is always a signed integer type. For integer execution types, extra precision is provided within the hardware, including the accumulators, so that conversions from unsigned to signed do not affect instruction correctness.

Register Region Restrictions

A register region is described as *packed* if its elements are adjacent in memory, with no intervening space, no overlap, and no replicated values. If there is more than one element in a row, elements must be adjacent. If there is more than one row, rows must be adjacent. When two registers are used, the registers must be adjacent and both must exist.

The following register region rules apply to the GEN implementation.

1. **General Restrictions Based on Operand Types**

   There are these general restrictions based on operand types:

   1. Where *n* is the largest element size in bytes for any source or destination operand type, *ExecSize* * *n* must be <= 64.
   2. When the Execution Data Type is wider than the destination data type, the destination must be aligned as required by the wider execution data type and specify a *HorzStride* equal to the ratio in sizes of the two data types. For example, a *mov* with a D source and B destination must use a 4-byte aligned destination and a *Dst.HorzStride* of 4.

2. **General Restrictions on Regioning Parameters**

   The mapping of data elements within the region of a source operand is in row-major order and is determined by the region description of the source operand, the destination operand, and the *ExecSize*, with these restrictions:

   1. *ExecSize* must be greater than or equal to *Width*.
   2. If *ExecSize* = *Width* and *HorzStride* ≠ 0, *VertStride* must be set to *Width* * *HorzStride*.
   3. If *ExecSize* = *Width* and *HorzStride* = 0, there is no restriction on *VertStride*.
   4. If *Width* = 1, *HorzStride* must be 0 regardless of the values of *ExecSize* and *VertStride*.
   5. If *ExecSize* = *Width* = 1, both *VertStride* and *HorzStride* must be 0.
   6. If *VertStride* = *HorzStride* = 0, *Width* must be 1 regardless of the value of *ExecSize*.
   7. *Dst.HorzStride* must not be 0.
   8. *VertStride* must be used to cross GRF register boundaries. This rule implies that elements within a '*Width*' cannot cross GRF boundaries.

3. **Region Alignment Rules for Direct Register Addressing**
   1. In Direct Addressing mode, a source cannot span more than 2 adjacent GRF registers.
   2. A destination cannot span more than 2 adjacent GRF registers.
   3. When a source or destination spans two registers, there are restrictions that vary by project, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Source or Destination Spans Two Registers

Restriction
When an instruction has a source region spanning two registers and a destination region contained in one register, the number of elements must be the same between two sources and one of the following must be true: 1. The destination region is entirely contained in the lower OWord of a register. 2. The destination region is entirely contained in the upper OWord of a register. 3. The destination elements are evenly split between the two OWords of a register.

When destination spans two registers, the source may be one or two registers. The destination elements must be evenly split between the two registers.
`// Case (a) Two destination registers with one source registers.`
`mov (8) r10.4<1>:f r11.5<8;8,1>:w`
`// Case (b) Two destination registers with two source registers.`
`mov (8) r10.4<1>:f r11.12<8;8,1>:w`

4. **Special Cases for Byte Operations**
   1. When the destination type is byte (UB or B) only a 'raw move' using the *mov* instruction supports a packed byte destination register region: *Dst.HorzStride* = 1 and *Dst.DstType* = (UB or B). This packed byte destination register region is not allowed for any other instructions, including a 'raw move' using the *sel*instruction, because the *sel* instruction is based on Word or DWord wide execution channels.
   2. This relaxed alignment rule does not apply for this stepping.

      There is a relaxed alignment rule for byte destinations. When the destination type is byte (UB or B), destination data types can be aligned to either the lowest byte or the second lowest byte of the execution channel. For example, if one of the source operands is in word mode (a signed or unsigned word integer), the execution data type will be signed word integer. In this case the destination data bytes can be either all in the even byte locations or all in the odd byte locations.

      This rule has two implications illustrated by this example:

```
// Example:
mov (8) r10.0<2>:b r11.0<8;8,1>:w
mov (8) r10.1<2>:b r11.0<8;8,1>:w

// Dst.HorzStride must be 2 in the above example so that the destination
```

```
// subregisters are aligned to the execution data type, which is :w.
// However, the offset may be .0 or .1.
// This special handling applies to byte destinations ONLY.
```

5. **Special Cases for Word Operations**

   There are some special cases for word operations for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may not show and there are no special cases in this category.

There is a relaxed alignment rule for word destinations. When the destination type is word (UW, W, HF), destination data types can be aligned to either the lowest word or the second lowest word of the execution channel. This means the destination data words can be either all in the even word locations or all in the odd word locations.
`// Example:`
`add (8)  r10.0<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf`
`add (8)  r10.1<2>:hf r11.0<8;8,1>:f r12.0<8;8,1>:hf`
 	
`// Note: The destination offset may be .0 or .1 although the destination subregister`
`// is required to be aligned to execution datatype.`

6. **Special Requirements for Handling Double Precision Data Types**

   There are special requirements for handling double precision data types that vary by project, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Special Requirements for Handling Double Precision Data Types

Requirement

In Align1 mode, all regioning parameters must use the syntax of a pair of packed floats, including channel selects and channel enables.
`// Example:`
`mov (8) r10.0.xyzw:df r11.0.xyzw:df`
`// The above instruction moves four double floats. The .x picks the`
`// low 32 bits and the .y picks the high 32 bits of the double float.`

In Align1 mode, all regioning parameters like stride, execution size, and width are in units of element size. However in Align16 mode, the channel selects and channel enables must always be used in pairs of packed floats, because these parameters are defined for DWord elements ONLY.
`// Example:`
`mov (4) r10.0<1>:df r11.0<4;4,1>:df`
`// The above instruction moves four double floats.`

When source or destination datatype is 64b or operation is integer DWord multiply, regioning in Align1 must follow these rules: 1. Source and Destination horizontal stride must be aligned to the same qword.

Requirement

Example:
`// mov (4) r10.0:df r11.0<16;8,2>:f // Source stride must be 2 since datatype is smaller.`
`// mov (4) r10.0<2>:f r11.0<4;4,1>:df // Destination stride must be 2 since datatype is smaller.`
`// mul (4) r10.0<2>:d r11.0<8;4,2>:d`
2. Regioning must ensure Src.Vstride = Src.Width * Src.Hstride.
3. Source and Destination offset must be the same, except the case of scalar source.

When source or destination datatype is 64b or operation is integer DWord multiply, indirect addressing must not be used.
ARF registers must never be used with 64b datatype or when operation is integer DWord multiply.
If Align16 is required for an operation with QW destination and non-QW source datatypes, the execution size cannot exceed 2.
In Align16 mode, format conversion from double-float to floats is not allowed when source is immediate data
When source or destination datatype is 64b or operation is integer DWord multiply, DepCtrl must not be used.

7. **Special Requirements for Handling Mixed Mode Float Operations**

There are some special requirements for handling mixed mode float operations for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Requirement
In Align16 mode, when half float and float data types are mixed between source operands OR between source and destination operands, the register content are assumed to be packed. In such cases the execution size reflects the number of float elements. Since a stride of 1 is assumed, source is selected in packed form and 16 bit packed data is updated on the destination operand, if the datatype is half-float.

For Align16 mixed mode, both input and output packed f16 data must be oword aligned, no oword crossing in packed f16.
Examples:
`Case (a)`
`mad (8) r10.0.xy:hf r11.0.xxxx:f r12.xyzw:hf r13.yyyy:hf`
`// The 16b of each word (r12.0, r12.1, r12.2, r12.3.. and so on) forms the source operand.`
`// r13.1 and r13.5 is replicated for source operand.`
`// The lower 16b of a Dword is updated for destination. With channel enables .xy , r10.0, r10.1, r10.4 and r10.5 are updated.`
 	
`Case (b)`
`mad (8) r10.0.xy:f r11.0.xxxx:f r12.xyzw:hf r13.yyyy:hf`
`// The example is similar to Case(a), except that entire DWord is updated on the destination.`

In Align16 mode, replicate is supported and is coissueable. `mad(8) r20.xyzw:hf r3.0.r:f r6.0.xyzw:hf r6.0.xyzw:hf {Q1}`

Requirement

No SIMD16 in mixed mode when destination is packed f16 for both Align1 and Align16.
`mad(8) r3.xyzw:hf r4.xyzw:f r6.xyzw:hf r7.xyzw:hf`
`add(8) r20.0<1>:hf r3<8;8,1>:f r6.0<8;8,1>:hf {Q1}`

No accumulator read access for Align16 mixed float.
When source is float or half float from accumulator register and destination is half float with a stride of 1, the source must register aligned. i.e., source must have offset zero.

No swizzle is allowed when an accumulator is used as an implicit source or an explicit source in an instruction. i.e. when destination is half float with an implicit accumulator source, destination stride needs to be 2.
`mac(8) r3<2>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`
`mov(8) r3<1>:f acc0.0<8;4,2>:hf`

In Align16, vertical stride can never be zero for f16 `add(8) r3.xyzw:hf r4.0<4>xyzw:f r6.0<0>.xyzw:hf`

Math operations for mixed mode:
- In Align16, only packed format is supported
`math(8) r3.xyzw:hf r4.0.<4>xyzw:f r6.0<0>.xyzw:hf 0x09`
- In Align1, f16 inputs need to be strided
`math(8) r3<1>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`

In Align1, destination stride can be smaller than execution type. When destination is stride of 1, 16 bit packed data is updated on the destination. However, output packed f16 data must be oword aligned, no oword crossing in packed f16. `add(8) r3<1>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`
When packed f16 is used as destination datatype, the subregister MUST be 0.
No SIMD16 in mixed mode when destination is f32. Instruction Execution size must be no more than 8.
Indirect Addressing on source is not supported when source and destination data types are mixed float.
No mixed mode select instruction with f16 packed destination. For example: `(f0.0) (8) r3<1>:hf r4.0<8;8,1>:f r6.0<8;4,2>:hf`

8. **Regioning Rules for Register Indirect Addressing**

Regioning rules for register indirect addressing vary for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Rules

1. When the execution size and destination regioning parameters require two adjacent registers, these registers are accessed using one index register ONLY.

```
// Example:
mov (16) r[a0.0]:f r10:f
// The above instruction behaves the same as the following two
instructions:
mov (8) r[a0.0]:f r10:f
mov (8) r[a0.0, 8*4]:f r11:f
```

2. When the destination requires two registers and the sources are 1x1 indirect mode, the sources must be assembled from two GRF registers accessed by a single index register. The data for each destination GRF register is entirely derived from one source register. This is ensured by appropriate use of regioning parameters. The exception to this is the use of indirect scalar sources, where the same element is used across the execution size.

```
// Example:
// Case (a)
add (16) r[a0.0]:f r[a0.2]:f r[a0.4]:f
// The above instruction behaves the same as the following two
instructions:
add (8) r[a0.0]:f r[a0.2]:f r[a0.4]:f
add (8) r[a0.0, 8*4]:f r[a0.2, 8*4]:f r[a0.4, 8*4]:f
// Note that the immediate for the second instruction is based on
regioning.
// In this case, it is 8 DWs.

// Case (b)
add (16) r[a0.0]:ud r[a0.2]<4;8,1>:w r10<8;8,1>:ud
// The above instruction behaves the same as the following two
instructions:
add (8) r[a0.0]:f r[a0.2]<4;8,1>:w r10<8;8,1>:ud
add (8) r[a0.0, 8*4]:f r[a0.2, 4*2]<4;8,1>:w r11<8;8,1>:ud
// Note that the immediate for the second instruction is based on
regioning.
// VertStride of 4 with data type of word.

// Case (c):
add (16) r[a0.0]:f r[a0.2]:f r[a0.4]<0;1,0>:f
// The above instruction behaves the same as the following two
instructions:
add (8) r[a0.0]:f r[a0.2]:f r[a0.4]<0;1,0>:f
add (8) r[a0.0, 8*4]:f r[a0.2, 8*4]:f r[a0.4]<0;1,0>:f
// Note that the src1 indirect address does not change.
```

3. Indirect addressing on src1 must be a 1x1 indexed region mode.
4. When a Vx1 or a VxH addressing mode is used on src0, the destination may use one or two registers.

```
// Example:
// Case (a)
add (16) r[a0.0]<1>:d r[a0.0]<4,1>:ud r16.0<8;8,1>:ud
// The above instruction behaves the same as the following two
instructions:
add (8) r[a0.0]<1>:d r[a0.0]<4,1>:ud r16.0<8;8,1>:ud
add (8) r[a0.0, 8*4]<1>:d r[a0.2]<4,1>:ud r17.0<8;8,1>:ud
// Since the pointer (index register) is incremented every 4 elements
// (width), the second instruction moves from a0.0 to a0.2.
```

Rules

// Case (b)
add (16) r10.0<2>:uw r[a0.0, 0]<1,0>:uw r16.0<8;8,1>:uw
// The above instruction behaves the same as the following two instructions:
add (8) r10.0<2>:uw r[a0.0, 0]<1,0>:uw r16.0<8;8,1>:uw
add (8) r11.0<2>:uw r[a0.8, 0]<1,0>:uw r17.0<8;8,1>:uw
// Since the pointer (index register) is incremented every 1 element
// (width), the second instruction moves from a0.0 to a0.8.
 	
5. Indirect addressing on the destination must be a 1x1 indexed region mode.

Execution size of 32 is NOT supported in Vx1 or VxH modes.

1. **Special Restrictions**

   There are some special restrictions on register region access for specific projects, described in the following table. If you are viewing a version of the BSpec limited to other particular projects, the table may appear with no data rows.

Restriction

In Align16 mode, the channel selects and channel enables apply to a pair of half-floats, because these parameters are defined for DWord elements ONLY. This is applicable when both source and destination are half-floats.
// Example:
mad (8) r10.0.xy:hf r11.0.xxxx:hf r12.0.xyzw:hf r13.xyzw:hf
// The .xxxx on the channel select implies the pair of half-floats, r11.0 and r11.1 is replicated.
// The .xy on the channel enable implies that 2 DWs or 4 half-floats r10.0, r10.1, r10.2 and r10.3 are updated.

Ternary instruction with condition modifiers must not use SIMD32.
The mul/mach macro sequence MUST not use Align16 Access Mode.
All flow control (branching) instructions must use the Align1 access mode.
When destination regioning spans two general registers and execution size is less than 8: 1. The flag registers cannot be used for implicit update (by condition modifiers). 2. The flag registers cannot be used in a select operation (opcode SEL).

Restriction
A POW/FDIV operation must not be followed by an instruction that requires two destination registers.
When using Align16 mode for conversion of data elements of different sizes, both source and destination must be one register each.

In Align16 mode, each destination register gets all data from one source register. This means, the data for one destination register is never scattered across two source registers.
`// Example:`
`// Allowed – all sources are contained within one register.`
`mul (8) r10.0:f r11.0:f r12.4<0>:f`
 	
`// NOT Allowed – src1 (r14) is scattered across two registers.`
`mad (8) r10.0:f r12.0<0>:f r14.4:f r16.0:f`

Conversion between Integer and HF (Half Float) must be DWord-aligned and strided by a DWord on the destination.
`// Example:`
`add (8) r10.0<2>:hf r11.0<8;8,1>:w r12.0<8;8,1>:w`
`// Destination stride must be 2.`
`mov (8) r10.0<2>:w r11.0<8;8,1>:hf`
`// Destination stride must be 2.`

Conversion between Integer and HF (Half Float) must be DWord-aligned and strided by a DWord on the destination.
For CHV:A, such format conversion is not allowed in Align16 mode.
`// Example:`
`add (8) r10.0<2>:hf r11.0<8;8,1>:w r12.0<8;8,1>:w`
`// Destination stride must be 2.`
`mov (8) r10.0<2>:w r11.0<8;8,1>:hf`
`// Destination stride must be 2.`

When Source is vector immediate, destination must NOT be a byte datatype.
When destination spans two registers, if the execution size is equal to width, horizontal stride

Restriction
must NOT be 0.
Any instruction with indirect addressing using vx1 or vxh must use {NoDDCLr}
The src, dst overlapping behavior with the second half src and the first half destination to the same register must not be used with any compressed instruction.

Destination Operand Description

This topic is currently under development.

Destination Region Parameters

Based on the above restrictions, a subset of register region parameters are sufficient to describe the destination operand:

- Destination Register Origin
  - Destination Register Number and Destination Subregister Number for direct register addressing mode
  - A Scalar Destination Register Index for register-indirect-register addressing mode
- Destination Register ‘Region’ – Note that destination register region does not have full region description parameters
  - Destination Horizontal Stride

SIMD Execution Control

This section of the BSpec discusses SIMD execution, both with and without predication. See the subtopics for more details.

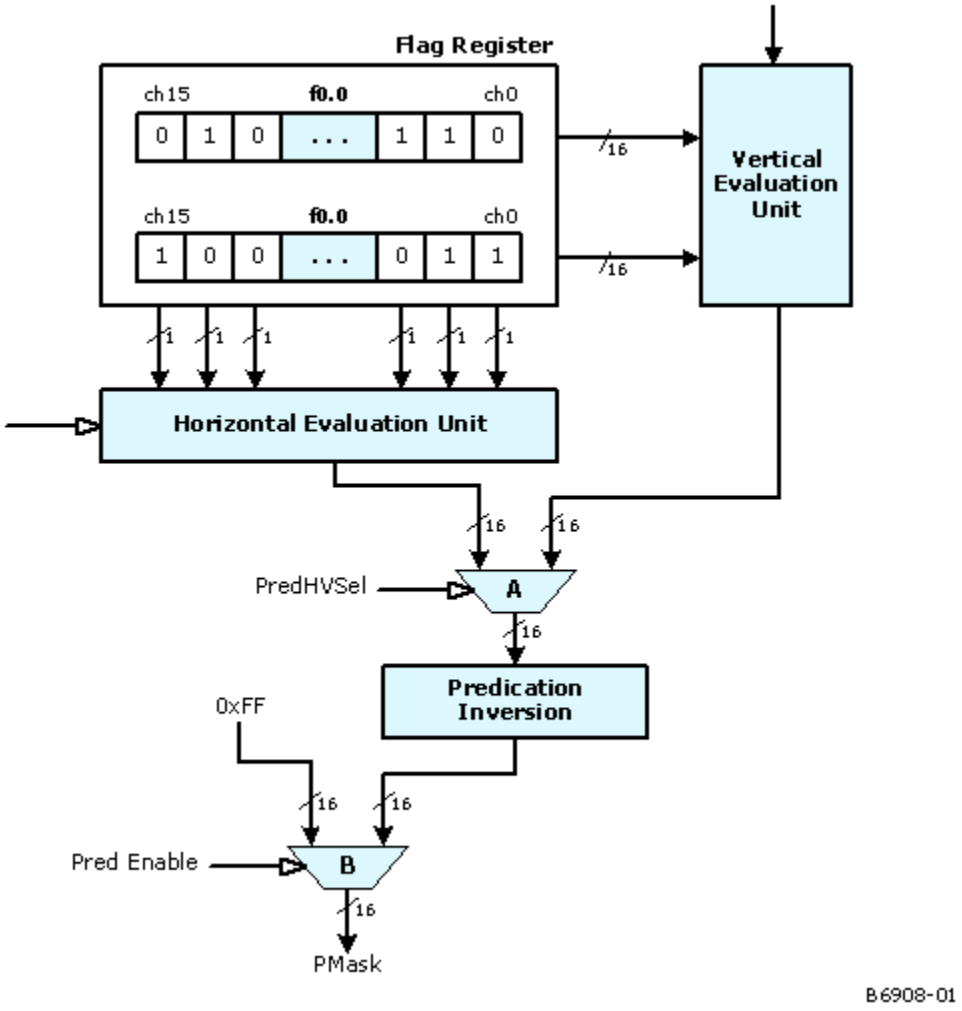
Predication

Predication is the conditional SIMD channel selection for execution on a per instruction basis. It is an efficient way of dynamic SIMD channel enabling without paying branch instruction overhead. When predication is enabled for an instruction, a Predicate Mask (PMask), which contains 16-bit channel enables, is generated internally in EU. Note that PMask is not a software visible register. It is provided here to explain how SIMD execution control works. PMask generation is based on the Predication Control (*PredCtrl*) field, Predication Inversion (*PredInv*) field and the flag source register in the instruction word. See Instruction Summary chapter for definition of these fields.

*Predication*shows the block diagram of the hardware logic to generate PMask. PMask is generated based on combinatory logic operation of the bits in the flag register. Instruction field *PredCtrl* controls the horizontal evaluation unit and vertical evaluation unit. MUX A in the figure selects whether horizontally-evaluated results or vertically-evaluated results are sent to the Predication Invertion unit. The *PredInv* field controls the Prediction Inversion unit. Either one 16-bit flag subregister or the whole flag register may be selected to generate the PMask depending on the predication control modes. MUX B indicates that predication can be enabled and disabled. Predication can be grouped into the following three categories. Predication functionality also depends on the Access Mode of the instruction.

- No predication: Of course, predication can be disabled. This is the most commonly used case.
- Predication with horizontal combination: the predicate mask is generated based on combinatory logic operation of bits within a selected flag subregister.
- Predication with vertical combination: the predicate mask is generated based on combinatory logic operation of bits across flag multiple subregisters.

Generation of predication mask



No Predication

When PredCtrl field of a given instruction is set to 0 (“no predication”), it indicates that no predication is applied to this instruction. Effectively, the resulting PMask is all 1’s. This is shown by the 2:1 multiplexer B controlled by the Pred Enable signal in *Predication*. Where predication is not enabled for an instruction, multiplex B is selected to output 0xFF to PMask.

Predication with Horizontal Combination

Predication with horizontal combination inputs the 16 bits of a single flag subregister (f0.0:uw or f0.1:uw) and passes them through combinatory logic of the Horizontal Evaluation unit to create PMask.

The simplest combination is ‘no combination’ – the same 16 bits from selected flag subregister are output to MUX A. In this case, a bit in the selected flag subregister controls the conditional execution of the corresponding execution channel. Let the selected flag subregister be denoted as f0.#, the following pseudo code describes the predicate mask generation for predication with sequential flag channel mapping.

```
    If (PredCtrl == “Sequential flag channel mapping”) {
For (ch=0; ch<16; ch++)
PMask[ch] = (PredInv == TRUE) ? ~f0.#[ch] : f0.#[ch];
    }
```

More complex horizontal evaluation is based on channel grouping. A group of adjacent channels (bits from flag subregister) are evaluated together and a single bit is replicated to the group. The size of groups is in power of 2. The supported combination depends on the Access Mode of an instruction.

In **Align16** access mode, horizontal combination is based on 4-channel groups.

- Channel replication: PredCtrl of ‘.x’, ‘.y’, ‘.z’ and ‘.w’ select a single channel from each 4-channel group and replicate it as the output for the group. For example, PredCtrl = ‘.x’ means that channel 0 in each group is replicated.
- OR combination: PredCtrl of ‘.any4h’ means that if **any** of the channel in a group is enabled, outputs for the 4 channels in the group are all enabled.
- AND combination: PredCtrl of ‘.all4h’ means that only when **all** of the channels in a group are enabled, the output for the group is enabled.

These combinations in **Align16** mode can be described by the following pseudo-code.

```
    If (Access Mode == Align16) {
For (ch = 0; ch < 16; ch += 4)
Switch (PredCtrl) {
Case '.x': bTmp = f0.#[ch]; break;
Case '.y': bTmp = f0.#[ch+1]; break;
Case '.z': bTmp = f0.#[ch+2]; break;
Case '.w': bTmp = f0.#[ch+3]; break;
Case '.any4h': bTmp = f0.#[ch] | f0.#[ch+1] | f0.#[ch+2] | f0.#[ch+3]; break;
Case '.all4h': bTmp = f0.#[ch] & f0.#[ch+1] & f0.#[ch+2] & f0.#[ch+3]; break;
}
bTmp = (PredInv == TRUE) ? ~bTmp : bTmp;
PMask[ch] = PMask[ch+1] = PMask[ch+2] = PMask[ch+3] = bTmp;
} }
```

In **Align1** access mode, horizontal combination is based on AND combination '.any#h' and OR combination '.all#h' on channel groups with various sizes, where # is the number of channels in a group ranging from 2 to 16. This is described by the following pseudo-code.

```
    If (Access Mode == Align1) {
Switch (PredCtrl) {
Case '.any2h': groupSize = 2; <op> = '|'; break;
Case '.all2h': groupSize = 2; <op> = '&'; break;
Case '.any4h': groupSize = 4; <op> = '|'; break;
Case '.all4h': groupSize = 4; <op> = '&'; break;
Case '.any8h': groupSize = 8; <op> = '|'; break;
Case '.all8h': groupSize = 8; <op> = '&'; break;
Case '.any16h': groupSize = 16; <op> = '|'; break;
Case '.all16h': groupSize = 16; <op> = '&'; break;
}
For (ch = 0; ch < 16; ch += groupSize) {
For (inc = 0, bTmp = FALSE; inc < groupSize; inc ++)
bTmp = bTmp <op> f0.#[ch+inc];
For (inc = 0; inc < groupSize; inc ++)
PMask[ch+inc] = bTmp;
} }
```

Predication with Vertical Combination

Predication with vertical combination uses both flag subregister as inputs. The AND or OR combination is across the subregisters on a channel by channel basis. This is shown by the following pseudo-code.

```
        If (Access Mode == Align1) {
For (ch = 0; ch < 16; ch ++) {
If (PredCtrl == 'any2v')
PMask[ch] = f0.0[ch] | f0.1[ch]
Else If (PredCtrl == 'any2h')
PMask[ch] = f0.0[ch] & f0.1[ch]
}
        }
```

End of Thread

There is no special instruction opcode (such as an END instruction) to cause the thread to terminate execution. Instead, the end of thread is signified by a *send* instruction with the end-of-thread (EOT) sideband bit set. Upon executing a *send* instruction with EOT set, the EU stops on the thread. Upon observing an EOT signal on the output message bus, the Thread Dispatcher makes the thread's resource available. If a thread uses pre-allocated resource managed by a fixed function, such as URB handles and scratch memory, some fixed function protocol also requires the thread to terminate with the message header phase to carry the information in order for the fixed function to release the pre-allocated resource.

EU hardware guarantees that if a terminated thread has in-flight read messages or loads at the time of 'end' that their writebacks will not interfere with either other threads in the system or new threads loaded in the system in the future.

More details can be found in the *send* instruction description in Instruction Reference chapter.

Assigning Conditional Flags

Instructions can output two sets of conditional signals, one set from before the outputs clamping/re-normalizing/format conversion logic, we call this the pre conditional signals. The second set is generated from the final results after clamping and re-normalizing/format conversion logic, and we call this the post conditional signals. The post conditional signals are used for fusing the DirectX compare instruction. **Note:** The flags generated from the post conditional signals should be equivalent to the flags generated by a separate *cmp* instruction after the current arithmetic instruction.

The pre conditional signals are used to generated flags for *cmp*/*cmpn* instructions only, this logically does the compare of the two input sources. The post conditional signals are used to generated flags for all the other arithmetic instructions, this logically does the compare of the result with zero.

cmpn with both sources as NaNs is a don't care case as this doesn't impact the MIN/MAX operations.

The pre conditional signals include the following:

- **pre_sign** bit: This bit reflects the sign of the computed result before going through any kind of clamping, normalizing, or format conversion logic.
- **pre_zero** bit: This bit reflects whether the computed result is zero before any kind of clamping, normalizing, or format conversion logic.

The post conditional signals include the following:

- **post_sign** bit: This bit reflects the sign of the final result after all the clamping, normalizing, or format conversion logic.
- **post_zero** bit: This bit reflects whether the final result is zero after all the clamping, normalizing, or format conversion logic.
- **OF** bit: This bit reflects whether an overflow occured in any of the computation of the current instruction, including clamping, re-normalizing, and format conversion.
- **NC** bit: The NaN computed bit indicates whether the computed result is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0.
- **NS0** bit: The NaN Source 0 bit indicates whether src0 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0.
- **NS1** bit: The NaN Source 1 bit indicates whether src1 of an execution channel is not a number. It carries valid information for instructions operating on floating point values. For an operation on integer operands, this bit is always 0. For an operation with one source operand, this bit is also set to 0. This bit is only used for the comparison instruction *cmpn*, which is specifically provided to emulate MIN/MAX operations. For any other instructions, this bit is undefined.
- Note that the bits generated at the output of a compute are before the **.sat**.

Flag Generation for *cmp* Instructions (The Supported Conditional Modifiers are .e, .ne, .g, .ge, .l, and .le.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
.e	Equal-to	**(pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether the two sources are equal. If either source is NaN (i.e. NC is true), the flag is forced to false.
.ne	Not-Equal-to	**! (pre_zero & ! (NS0 | NS1)).** This conditional modifier test whether the two sources are equal. It takes exactly the reverse polarity as the modifier **.e**.
.g	Greater-than	**(! pre_sign & ! pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is greater than src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.ge	Greater-than-or-equal-to	**((! pre_sign | pre_zero) & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is greater than or equal to src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.l	Less-than	**(pre_sign & !pre_zero & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is less than src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.le	Less-than-or-equal-to	**((pre_sign | pre_zero) & ! (NS0 | NS1)).** This conditional modifier tests whether src0 is less than or equal to src1. If either source is a NaN (i.e. NC is true), the flag is forced to false.

Flag Generation for All Instructions Other than *cmp/cmpn* Instructions (The Supported Conditional Modifiers are .e, .ne, .g, .ge, .l, .le, .o, and .u.)

Conditional Modifier	Meaning	Resulting Flag Value (for an execution channel)
.e	Equal-to	**(post_zero & ! NC).** This conditional modifier tests whether the result is equal to zero. If either source is NaN (i.e. NC is true), the flag is forced to false.
.ne	Not-Equal-to	**! (post_zero & ! NC).** This conditional modifier test whether the result is not equal to zero. It takes exactly the reverse polarity as modifier **.e**.
.g	Greater-than	**(! post_sign & ! post_zero & ! NC).** This conditional modifier tests whether result is greater than zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.ge	Greater-than-or-equal-to	**((! post_sign | post_zero) & ! NC).** This conditional modifier tests whether result is greater than or equal to zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.l	Less-than	**(post_sign & ! post_zero & ! NC).** This conditional modifier tests whether result is equal to zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.le	Less-than-or-equal-to	**((post_sign | post_zero) & ! NC).** This conditional modifier tests whether result is equal to or less than zero. If either source is a NaN (i.e. NC is true), the flag is forced to false.
.o	Overflow	**OF.** This conditional modifier tests whether the computed result causes overflow – the computed result is outside the range of the destination data type. Note: The legacy condition modifier behavior is different from IEEE exception Overflow flag. For inf float to int conversion, .o will set the legacy Overflow flag, but IEEE exception Overflow flag won't be set. All other internal conditional signals are ignored.
.u	Unordered	**NC.** This conditional modifier tests whether the computed result is a NaN (unordered). All other internal conditional signals are ignored.

Destination Hazard

GEN architecture has built-in hardware to avoid destination hazard.

Destination Hazard stands for the risk condition when multiple operations are trying to write to the same destination and the result of the destination may be ambiguous. This may or may not happen on GEN for two instructions with the same destination, or with destinations that have overlapped register region, depending on the ordering of the arrival of destination results. Let's consider two instructions in a thread with potential destination hazard. There may be other instruction between them as long as there is no instruction sourcing the same destination. Using register scoreboards, GEN hardware automatically takes care of the destination hazard by not issuing the second instruction until the destination scoreboard is cleared. However, for certain cases, in fact for most cases, such destination hazard indicated by the register scoreboard is false, causing unnecessary delay of instruction issuing. This may result in lower performance. The destination dependency control field in the instruction word {*NoDDClr* } allows software to selectively override such hardware destination dependency mechanism. Such performance optimization hooks must be used with extreme caution. When it is not certain that it is a false destination hazard, the programmer should rely on hardware to resolve the dependency.

As the destination dependency control field does not apply to *send* instruction, there is only one condition that a programmer may use the {*NoDDClr* } capability.

Description
Instructions other than send, may use this control as long as operations that have different pipeline latencies are not mixed. The operations that have longer latencies are: ▪ Opcodes *pln*, *lrp*, *dp**. ▪ Operations involving double precision computation. ▪ Integer DW multiplication where both source operands are DWs.

When a sequence of NoDDClr is used, the last instruction that completes the scoreboard clear must have a non-zero execution mask. This means, if any kind of predication can change the execution mask or channel enable of the last instruction, the optimization must be avoided. This is to avoid instructions being shot down the pipeline when no writes are required.

Example:

(f0.0) mov r10.0 r11.0 {NoDDClr}

(-f0.0) mov r10.0 r11.0 { NoDDClr}

In the above case, if predication can disable all writes to r10 for the second instructions, the instruction maybe shot down the pipeline resulting in un-deterministic behavior. Hence, This optimization must not be used in these cases.

Non-present Operands

Some instructions do not have two source operands and one destination operand. If an operand is not present for an instruction the operand field in the binary instruction must be filed with null. Otherwise, results are unpredictable.

Specifically, for instructions with a single source, it only uses the first source operand src0. In this case, the second source operand src1 must be set to null and also with the same type as the first source operand src0. It is a special case when src0 is an immediate, as an immediate src0 uses DW3 of the instruction word, which is normally used by src1. In this case, src1 must be programmed with register file ARF and the same data type as src0.

Instruction Prefetch

Due to prefetch of the instruction stream, the EUs may attempt to access up to 8 instructions (128 bytes) beyond the end of the kernel program – possibly into the next memory page. Although these instructions will not be executed, they must be accounted for the prefetch in order to avoid invalid page access faults. GFX software is required to pad the end of all kernel programs with 512b data. A more efficient approach would be to ensure that the page after all kernel programs is at least valid (even if mapped to a dummy page). Note that the **General State Access Upper Bound** field of the STATE_BASE_ADDRESS command can be used to prevent memory accesses past the end of the General State heap (where kernel programs must reside).

ISA Introduction

This chapter contains these sections that introduce this volume.

- Introducing the Execution Unit
- EU Terms and Acronyms
- EU Changes by Processor Generation
- EU Notation

Subsequent chapters cover:

- EU Data Types
- Execution Environment
- Exceptions
- Instruction Set Summary
- Instruction Set Reference
- EU Programming Guide

The EU Programming Guide provides some useful examples and information but is not a complete or comprehensive programming guide.

Introducing the Execution Unit

This section introduces the Execution Unit (EU), a simple and capable processor within the GPU that supports graphics processing within the graphics pipelines, can do general purpose computing (GPGPU), and responds to exceptional conditions via the System Routine.

The EU provides parallelism at two levels: thread and data element. Multiple threads can execute on the EU; the number executing concurrently depends on the processor and is transparent to EU code. Each thread has its own registers (GRF and ARF, described below). Most EU instructions operate on arrays of data elements; the number of data elements is normally the *ExecSize (execution size)* or number of channels for the instruction. A *channel* is a logical unit of execution for data element access, masking, and flow control within instructions. The number of channels is independent of the number of physical ALUs or FPUs for a particular graphics processor.

EU native instructions are 128 bits (16 bytes) wide. Some combinations of instruction options can use compact instruction formats that are 64 bits (8 bytes) wide. Identifying instructions that can be compacted and creating the compact representations is done by software tools, including compilers and assemblers.

Data manipulation instructions have a destination operand (dst) and one, two, or three source operands (src0, src1, or src2). The instruction opcode determines the number of source operands. An instruction's last source operand can be an immediate value rather than a register.

Data read or written by a thread is generally in the thread's *GRF (General Register File)*, 128 general registers, each 32 bytes. A data element address within the GRF is denoted by a register number (r0 to r127) and a subregister number. In the instruction syntax, subregister numbers are in units of data element size. For example, a :d (Signed Doubleword Integer) element can be in subregister 0 to 7, corresponding to byte numbers in the instruction encoding of 0, 4, ... 28.

The EU cannot directly read or write data in system memory.

Specialized registers used to implement the ISA are in a distinct per thread *Architecture Register File (ARF)*. Each such register or group of related registers has its own distinct name. For example, ip is the instruction pointer and f0 is a flags register. An ARF register can be a src0 or dst operand but not a src1 or src2 operand. There are restrictions on how particular ARF registers are accessed that should be understood before directly reading or writing those registers. See the ARF Registers section for more information.

The EU supports both integer and floating-point data types, as described in the Numeric Data Types section.

For EU flow control, each channel has its own per-channel instruction pointer (PcIP[n]) and only executes an instruction when IP == PcIP[n] and any other masks enable the channel. Most flow control instructions use signed offsets from the current instruction address to reference their targets. Unconditional branches are done using *mov* with IP as the destination. Flow control can also use SPF (Single Program Flow) mode to execute with a single instruction pointer (IP).

The EU ISA supports predication, masking, regioning, swizzling, some type conversions, source modification, saturation, accumulator updates, and flag updates as part of instruction execution:

- *Predication* creates a bit mask (*PMask*) to enable or disable channels for a particular instruction execution. Pmask is derived from flag register and subregister values using boolean formulas determined by the PredCtrl (Predicate Control) and PredInv (Predicate Inversion) instruction fields. See the Predication section.
- *Masking* is the overall process of determining which channels execute for a given instruction based on five factors:
  - Number of channels (only channels in [0, *ExecSize* - 1] can execute)
  - Execution mask (*EMask*)
  - Whether the channel is on the instruction (if not in Single Program Flow mode and MaskCtrl is not NoMask)
  - Predicate mask (*PMask*)
  - In Align16 mode, any enabling of channels using the Dst.ChanEn instruction field (if MaskCtrl is not NoMask).
- *Regioning* specifies an array of data elements contained in one or two registers, with options for scattering, interleaving, or repeating data elements in registers using width and stride values, subject to significant constraints. Regioning also includes access mode (Align1 or Align16) and addressing mode (Direct or Indirect). See the Registers and Register Regions section.
- *Swizzling* allows small scale reordering of data elements within groups of four at the input using the modulo 4 channel names x, y, z, and w. For example, a swizzle of .wzyx with an *ExecSize* of 8 reads execution channels 0 to 7 from these input channels: 3, 2, 1, 0, 7, 6, 5, and 4. Swizzling is only available in the Align16 access mode, described in the Execution Environment chapter.
- *Type Conversions* do any needed conversion from source data type to execution data type and from execution data type to destination data type. See Execution Data Type for more information. Each instruction description indicates what combinations of data types are supported.
- *Source Modification* modifies a source operand just before doing the requested operation. For a numeric operation, the choices are:
  - No modification (normal).
  - - indicating negation.
  - (abs) indicating absolute value.
  - -(abs) indicating a forced negative value.

  Source modification logically occurs after any conversion from source data type to execution data type. Each instruction description indicates whether it supports source modification.
- *Saturation* clamps result values to the nearest value within a saturation range determined by the destination type. For a floating-point type, the saturation range is [0.0, 1.0]. For an integer type, the saturation range is the entire range for that type, for example [0, 65535] for the UW (Unsigned Word) type. Each instruction description indicates whether it supports saturation.
- *Accumulator Updates* optionally update the accumulator register or registers in the ARF with destination values as a side effect of instruction execution. The AccWrCtrl instruction field enables accumulator updates. The Accumulator Disable flag in control register 0 (cr0) can be used to

disable accumulator updates, regardless of AccWrCtrl values; for example, this flag may be used in the System Routine.

- *Flag Updates* optionally update a flags register and subregister (f0.0, f0.1, f1.0, or f1.1) with conditional flags based on the CondModifier (Condition Modifier) instruction field. For example, a CondModifier of **.nz** (not zero) assigns flag bits based on whether result elements are not zero (1) or zero (0). Each instruction description indicates whether it supports the Condition Modifier and any restrictions on the values supported.

The EU is not required to execute steps in its internal pipeline sequentially or in order, so long as it produces correct results.

The assembler syntax uses spaces between operands and encloses ExecSize and any predicate in parentheses. Instruction mnemonics, register names, conditional modifiers, predicate controls, and type designators use lowercase. Function names used with the math instruction are UPPERCASE.

```
( pred ) inst cmod sat ( exec_size ) dst src0 src1 { inst_opt, ... }
```

General register destination regions use the syntax r*m.n*<*HorzStride*>:type. General register directly addressed source regions use the syntax r*m.n*<*VertStride*;*Width*,*HorzStride*>:type. You need to understand more about register regioning to understand all of these terms.

The following example assembly language instruction adds two packed 16-element single-precision Float arrays in r4/r5 and r2/r3 writing results to r0/r1, only on those channels enabled by the predicate in f0.0 along with any other applicable masks.

```
(f0.0) add (16) r0.0<1>:f r2.0<8;8,1>:f r4.0<8;8,1>:f
```

EU Terms and Acronyms

This section provides three tables describing EU general terms and acronyms, EU data types, and EU selected ARF registers.

EU General Terms and Acronyms

Term	Description
ALT mode	A floating-point execution mode that maps +/- inf to +/- fmax, +/- denorm to +/-0, and NaN to +0 at the FPU inputs and never produces infinities, denormals, or NaN values as outputs. See IEEE mode.
ALU	Arithmetic Logic Unit. A functional block that performs integer arithmetic and logic operations, as distinct from instruction fetch and decode, floating-point operations (see FPU), or messaging.
AOS	Array Of Structures. Also see SOA.
ARF	Architecture Register File, a distinct register file containing registers used to implement specific ISA features. For example the Instruction Pointer and condition flags are in ARF registers. See GRF.
byte	An 8-bit value aligned on an 8-bit boundary and the basic unit of addressing. Bits within a byte are denoted 0 to 7 from LSB to MSB.
channel	A logical unit of SIMD data parallel execution within a thread and within the EU. The number of physical ALUs or FPUs is not directly related to the number of channels. CHV, BSW supports up to 32 channels.
compact instruction	A 64-bit instruction encoded as described in the EU Compact Instructions section. Only some combinations of instruction parameters can be encoded as compact instructions. See native instruction.
compressed instruction	An instruction that writes to two destination registers. For example a SIMD16 instruction with Float operands can write channels 0 to 7 to one 32-byte general register and channels 8 to 15 to a second, consecutive 32-byte general register.
denorm	A very small but nonzero number in IEEE mode, with a magnitude less than the smallest normalized floating-point number representable in a particular floating-point format. Denormals lose precision as their values approach zero, called *gradual underflow*.
DWord	Doubleword. A 32-bit (4-byte) value aligned on a 32-bit (4-byte) boundary. Bits within a DWord are denoted 0 to 31 from LSB to MSB.
EOT	End of Thread. A flag set on a *send* or *sendc* instruction to terminate a thread's execution on the EU.
EU	Execution Unit. The single GPU unit described in this volume. This volume describes individual data parallel execution paths within a thread in the EU as *channels*. A few fields, like EUID, use EU to refer to a particular hardware resource used to implement the overall EU.
exception	An error or interrupt condition that arises during execution that may transfer control to the System Routine. Some exceptions can be disabled, preventing such transfers. As defined in this volume, some errors do not produce exceptions.
ExecSize	The number of execution channels for a particular instruction. Channels within that number are enabled or disabled by various masks.

Term	Description
floating-point	Numeric types that allow fractional values and often a wider range than integer types. The EU supports binary floating-point types including the single precision type and the double precision type defined by the IEEE 754 standard.
GEN	GEN is sometimes used to refer to Intel's mainstream GPU architecture integrated with recent CPU generations.
GRF	General Register File, a distinct register file containing 128 general registers, r0 to r127. Each general register is 256 bits (32 bytes), can contain any type of data, and can be accessed with any valid combination of addressing mode, access mode, and region parameters. A general register is directly addressed using a register number and subregister number, or indirectly addressed using an address subregister (index register) and an address immediate offset.
IEEE mode	A floating-point execution mode that supports all the kinds of floating-point values described by the IEEE 754 standard: normalized finite nonzero binary floating-point numbers, signed zeros, signed infinities, signed denormals that are closer to zero than any normalized value but still nonzero, and NaN (not a number) values. See ALT mode.
index register	An address subregister when used for indirect addressing.
inf	Infinity, +inf or -inf, as a floating-point value in IEEE mode.
instruction	In this volume, *instruction* always refers to an EU instruction.
ISA	Instruction Set Architecture, processor aspects visible to programs and programmers and independent of a particular implementation, including data types, registers, memory access, addressing modes, exceptions, instruction encodings, and the instruction set itself. An ISA does not include instruction timing, hardware pipeline details, or the number of physical resources (ALUs, FPUs, instruction decoders) mapped to logical constructs (threads, channels). This volume also includes a recommended assembly language syntax, closely related to the ISA but logically distinct from it.
LSB	Least significant bit.
message	A data structure transmitted from a thread to another thread, to a shared function, or to a fixed function. Message passing is the primary communication mechanism of the GEN architecture.
MSB	Most significant bit.
NaN	Not a Number. A non-numeric value allowed in the standard single precision and double precision floating-point number formats. Quiet NaNs propagate through calculations and signaling NaNs cause exceptions. NaNs are not used in the ALT floating-point mode.
native instruction	A 128-bit instruction, the regular instruction format that allows all defined instruction parameters and options. Some instructions can also be encoded using a 64-bit compact instruction format.
OWord	Octword. A 128-bit (16-byte) value aligned on a 128-bit (16-byte) boundary. Bits within an OWord are denoted 0 to 127 from LSB to MSB. This term is used rarely and may be dropped from future versions of this volume.
packed	A register region is described as *packed* if its elements are adjacent in memory, with no intervening space, no overlap, and no replicated values. If there is more than one element in a row, elements must be adjacent. If there is more than one row, rows must be adjacent. When two registers are used, the registers must be adjacent and both must exist. The immediate vector data types are all described as *Packed* because each such type packs several

Term	Description
	small data elements into a 32-bit immediate value.
QWord	Quadword. A 64-bit (8-byte) value aligned on a 64-bit (8-byte) boundary. Bits within a QWord are denoted 0 to 63 from LSB to MSB.
region	A collection of data locations in registers and subregisters for a source or destination operand. The associated regioning parameters allow regions to be arrays with various layouts.
register	Part of the directly accessible state of an EU program, such as a general register in the GRF or an architecture register in the ARF. Note that system memory is not directly accessible.
SIMD	Single Instruction Multiple Data. Each EU instruction can operate on multiple data elements in parallel, as specified by the instruction's ExecSize.
SIP	System Instruction Pointer, the starting IP value for the System Routine.
SOA	Structure of Arrays. Also see AOS.
SPF	Single Program Flow. A mode in which every execution channel uses the common instruction pointer, IP in the ip register. The SPF bit in the control register is 1 to enable SPF and 0 to disable it. If SPF is disabled, then each execution channel n has its own instruction pointer, PcIP[n] and each channel n is only eligible to execute, subject to other masking, when PcIP[n] == IP.
swizzle	Rearrange data elements within a vector. The EU supports modulo four swizzling of register source operands at the input in the Align16 access mode.
System Routine	A global EU exception handling routine. Any enabled exception from any EU thread transfers control to this routine.
thread	An instance of a program executing on the EU. The life cycle for a thread on the EU starts with the first instruction after being dispatched to the EU by the Thread Dispatcher and ends after executing a *send* or *sendc* instruction with EOT set, signaling thread termination. Threads can be independent or can communicate with each other via the Message Gateway shared function.
word	A 16-bit (2-byte) value aligned on a 16-bit (2-byte) boundary. Bits within a word are denoted 0 to 15 from LSB to MSB. *Word* has denoted a 16-bit unit for Intel processors since the 8086 and 8088 processors were introduced in 1978.

The next table lists all EU numeric data types. See the Numeric Data Types section for more information about each data type.

EU Numeric Data Types (Listed Alphabetically by Short Name)

Short Name	Assembler Syntax	Long Name	Size in Bytes	Size in Bits	Integral or Float	Description
B	:b	Signed Byte Integer	1	8	I	Signed integer in the range -128 to 127.
D	:d	Signed Doubleword Integer	4	32	I	Signed integer in the range -2^{31} to $2^{31} - 1$.
DF	:df	Double Float	8	64	F	CHV, BSW: Double precision floating-point number.
F	:f	Float	4	32	F	Single precision floating-point number.
HF	:hf	Half Float	2	16	F	CHV, BSW: Half precision floating-point number.
Q	:q	Signed Quadword Integer	8	64	I	CHV, BSW: Signed integer in the range -2^{63} to $2^{63} - 1$.
UB	:ub	Unsigned Byte Integer	1	8	I	Unsigned integer in the range 0 to 255.
UD	:ud	Unsigned Doubleword Integer	4	32	I	Unsigned integer in the range 0 to $2^{32} - 1$.
UQ	:uq	Unsigned Quadword Integer	8	64	I	CHV, BSW: Unsigned integer in the range 0 to $2^{64} - 1$.
UV	:uv	Packed Unsigned Half Byte Integer Vector	4	32	I	Eight 4-bit unsigned integer values each in the range 0 to 15. Only used as an immediate value.
UW	:uw	Unsigned Word Integer	2	16	I	Unsigned integer in the range 0 to 65,535.
V	:v	Packed Signed Half Byte Integer Vector	4	32	I	Eight 4-bit signed integer values each in the range -8 to 7. Only used as an immediate value.
VF	:vf	Packed Restricted Float Vector	4	32	F	Four 8-bit restricted float values. Only used as an immediate value.
W	:w	Signed Word Integer	2	16	I	Signed integer in the range -32,768 to 32,767.

The next table lists the seven ARF registers that you should understand first, omitting several others. See the ARF Registers section for more information, including descriptions of additional registers not listed below.

EU Selected ARF Registers (Listed Alphabetically by Name)

Name	Assembler Syntax	Description
Accumulators	acc0, acc1	Data registers that can hold integer or floating-point values of various sizes. Many instructions can implicitly update accumulators with a copy of destination values, done by setting the AccWrCtrl instruction option. A few instructions, like *mac* (Multiply Accumulate), use the accumulators as an implicit source operand, useful for some iterative calculations. For CHV, BSW there are added accumulator registers acc2 to acc9 for special purposes; these added accumulators are not generally used.
Address Register	a0.*s*	Holds subregisters primarily used for indirect addressing. Each subregister is a 16-bit UW (Unsigned Word) value. For an indirectly addressed operand or element, the subregister value plus an AddrImm signed offset field determines the byte address (RegNum and SubRegNum) within the register file (GRF). For CHV, BSW there are 16 address subregisters.
Control Register	cr0.*s*	Contains bit fields for floating-point modes, flow control modes, and exception enable/disable. Also contains exception indicator flags and saves the AIP (Application Instruction Pointer) on transferring control to the System Routine to handle an exception.
Flags	f*r.s*	Used as the outputs for various channel conditional signals, such as equality/zero or overflow. Used as the inputs for predication. There are two 32-bit flags registers each containing two 16-bit subregisters.
Instruction Pointer (IP)	ip	References the current instruction in memory, as an unsigned offset from the General State Base Address. IP is the thread's overall instruction pointer. Each channel n can have its own instruction pointer (PcIP[n]). If not in Single Program Flow mode (SPF is 0) then only those channels where PcIP[n] == IP are eligible to execute the instruction, if enabled by all other applicable masks.
Null Register	null	Indicates a non-existent operand. Unused operands in the instruction format, like the unused second source operand field in a *mov* instruction, are encoded as null. For present source operands, reading a null source operand returns undefined values. For null destination operands, results are discarded but any implicit updates to accumulators or flags still occur.
State Register	sr0.*s*	Contains thread identification and scheduling fields, and mask fields for enabling or disabling channels.

Execution Units (EUs)

Each EU is a vector machine capable of performing a given operation on as many as 16 pieces of data of the same type in parallel (though not necessarily on the same instant in time). In addition, each EU can support a number of execution contexts called *threads* that are used to avoid stalling the EU during a high-latency operation (external to the EU) by providing an opportunity for the EU to switch to a completely different workload with minimal latency while waiting for the high-latency operation to complete.

For example, if a program executing on an EU requires a texture read by the sampling engine, the EU may not necessarily idle while the data is fetched from memory, arranged, filtered and returned to the EU. Instead the EU will likely switch execution to another (unrelated) thread associated with that EU. If that thread encounters a stall, the EU may switch to yet another thread and so on. Once the Sampler result arrives back at the EU, the EU can switch back to the original thread and use the returned data as it continues execution of that thread.

The fact that there are multiple EU cores each with multiple threads can generally be ignored by software. There are some exceptions to this rule: e.g., for:

Description
thread-to-thread communication (see *Message Gateway*, *Media*)
synchronization of thread output to memory buffers (see *Geometry Shader*)

In contrast, the internal SIMD aspects of the EU are very much exposed to software.

This volume will not deal with the details of the EUs.

EU Changes by Processor Generation

This section describes how the EU changes for particular processor generations. Instruction compaction tables can differ for each generation, so that is not mentioned in these lists. Particular readers and audiences can see only certain content in this section. Workarounds for particular generations, SKUs, or steppings are not included in these lists. Some small changes in instruction layouts are not included in these lists.

Description
These features or behaviors are added for CHV, BSW, continuing to later generations: • The maximum *ExecSize* increases to 32, for byte or word operands. • Increase the number of flag registers from one to two. • Add the NibCtrl field, used with QtrCtrl to select groups of channels or flags. • Add the DF (Double Float) data type, the first time an 8-byte data type is supported. DF only supports the IEEE floating-point mode and not the ALT floating-point mode. • Add a shared source data type field and a destination data type field for instructions with three source operands, allowing F (Float), DF (Double Float), D (Signed Doubleword Integer), or UD (Unsigned Doubleword Integer) types to be specified. • Add bit manipulation instructions: *bfi1*, *bfi2*, *bfrev*, *cbit*, *fbh*, and *fbl*. • Add the integer *addc* (Add with Carry) and *subb* (Subtract with Borrow) instructions. • Add the *brc* (Branch Converging) and *brd* (Branch Diverging) instructions. • For the *cmp* and *cmpn* instructions, relax the accumulator restrictions. • For the *sel* instruction, remove the accumulator restriction. • Add the Rounding Mode and Double Precision Denorm Mode fields in Control Register 0.
These features or behaviors are added for CHV, BSW, continuing to later generations: • DF (Double Float) operands use an element size of 8. Regioning and channel parameters for the DF type are determined normally, in the same way as for other types. • Add the channel enable register, flow control registers, and stack pointer register in the ARF. • In the Control Register, add the Force Exception Status and Control, Context Save Status, and Context Restore Status bits. • Relative instruction offsets (JIP, UIP) are now 32-bit values in units of bytes (rather than 16-bit values using 8-byte units) for some instructions: *brc*, *brd*, *call*, and *jmpi*. • A *call* instruction can get the relative instruction offset (JIP) from a register. • Add the *calla* (Call Absolute) instruction. • A *mov* instruction with different source and destination types can now use conditional modifiers.
These features or behaviors are added for CHV, BSW, continuing to later generations: • Add the HF (Half Float) type and a corresponding HF execution data type and execution path. • Add flags to indicate IEEE floating-point exceptions and to enable or disable exception reporting to those

Description
flags. • Add the Single Precision Denorm Mode bit in Control Register 0. It can be enabled to allow calculations using the F (Float) type in IEEE floating-point mode to support denormals and gradual underflow. • Add the Q (Signed Quadword Integer) and UQ (Unsigned Quadword Integer) types. Integer source types cannot mix 64-bit and 8-bit operands. Some integer instructions (e.g., *avg*) do not support Q or UQ source types. • Instructions with one source operand and a 64-bit source type can have immediate 64-bit source operands. • The JIP and UIP relative instruction offset fields in all remaining flow control instructions are 32-bit values in units of bytes (rather than 16-bit values using 8-byte units). • The instruction layout is noticeably different. The SrcType and DstType instruction fields are widened to allow for more type encodings as three types are added. The AddrSubRegNum instruction field is widened to allow for 16 address subregisters rather than 8. The layout now supports 64-bit immediate source operands for one-source instructions and 32-bit relative instruction offset fields for flow control instructions. • In the 3-source instruction format, widen the SrcType and DstType fields and add an encoding for the HF (Half Float) type. • Add a compact instruction format for 3-source instructions. • Use a different source modifier interpretation for logical (*and*, *not*, *or*, *xor*) instructions. • An accumulator source operand for a logical instruction can now have a source modifier. • Add eight address subregisters, increasing the number of address subregisters from 8 to 16. • For the *brc* and *brd* instructions do not allow the **Switch** instruction option. • For the *cmp* and *cmpn* instructions, remove the accumulator restrictions. • Add the *goto* instruction, reusing the opcode for the discontinued *fork* instruction. • Add the *join* instruction. • For the *lzd* instruction, remove the accumulator restriction. • The *mach* instruction reverses the roles of the two source operands compared to previous generations. • Add the *madm* instruction. • Enhance the *math* instruction to allow some immediate source values and support the INVM and RSQRTM functions. • For the *mul* instruction, relax the accumulator restriction on source operands so it applies for only integer source operands. • For the rounding instructions (*rndd*, *rnde*, *rndu*, and *rndz*), remove the accumulator restrictions. • Revise the *shl* and *shr* instructions to use the low 6 bits of the shift count in QWord mode, versus the low 5 bits otherwise. • Add the *smov* instruction. • Add eight accumulator registers, acc2 to acc9, used only for the special purpose of emulating IEEE-compliant fdiv and sqrt operations. • Add message control registers. • Widen the sp (Stack Pointer) register to 64 bits. • Add the IEEE Exception, Page Fault Status, and Page Fault Code bit fields in the State Register.

Description
• Remove some regioning restrictions when operands span two registers.
These features or behaviors are specific to CHV: • Calculations using the HF (Half Float) type do not support denormals or gradual underflow. • Revise the Slice ID and SubSlice ID (previously HalfSlice ID) bit fields in the State Register. • The System IP (SIP) is widened to 48 bits. However the EU still only uses the low 32 bits.

EU Notation

The Courier New font is used for code examples and for the Syntax, Format, and Pseudocode sections in the instruction reference.

The *italic* font style is used for instruction mnemonics outside of code (e.g., the *send* instruction), for syntactic production names, for key values in algorithms (*ExecSize*), and to emphasize a word or phrase. For example: When bit 10 is set, the destination register scoreboard is *not* cleared.

The **bold** font weight is used for the short name and long name of a bit field being described, for value names being defined, for syntactic terminals, for unnumbered subheadings, and for the terms Note or Workaround used to introduce a paragraph.

Bit field names and value names used where not being defined and not as syntactic terminals are in plain text. Bit field values in hex use the 0x prefix. The BSpec currently uses the 0x prefix for hex in some parts and the h suffix for hex in other parts. For single bits, values appear as simply 0 or 1. For multi-bit binary values, the appropriate number of binary digits appears with a b suffix.

Instruction mnemonics are lowercase. Function names invoked using the *math* instruction are UPPERCASE. For example, SQRT.

Device names in the new syntax do not use the Dev prefix or square brackets and often appear in tagged tables with a blue background. For example: Device names in the old syntax are in plain text in square brackets.

Tables describing bit field layouts or registers proceed from most significant to least significant bits. Figures showing bit fields or registers show most significant bits on the left and least significant bits on the right.

Any bit, field, or register described as Reserved should be regarded as undefined and unpredictable. Such bits should be treated as follows:

- When testing values, do not depend on the state of reserved bits. Mask out or otherwise ignore such bits.
- Sometimes software must initialize reserved bits. For example, a compiler must write complete instruction values when creating an instruction stream, including reserved bits. In such cases, write reserved bits as zeros unless otherwise indicated.
- Do not use reserved bits as extra storage for software-defined values; put nothing in such bits.
- When saving state and restoring state, save and restore any reserved bits as well.
- Do not assume that reserved bits are invariant between explicit writes. Software should function even if reserved bits change in undefined and unpredictable ways.

Any value, encoding, or combination of values or encodings described as Reserved must not be used. The EU's behavior is undefined in this case.

When a combination of instruction parameters or an EU state is described as producing undefined results or behavior, do not assume that undefined results or behavior are confined to specific instructions, operands, registers, or channels.

Execution Environment

This topic is currently under development.

EU Data Types

Fundamental Data Types

Numeric Data Types

Floating Point Modes

- IEEE Floating Point Mode
  - Partial Listing of Honored IEEE 754 Rules
  - Complete Listing of Deviations or Additional Requirements vs IEEE 754
  - Min/Max of Floating Point Numbers
- Alternative Floating Point Mode

Floating-Point Support [CHV, BSW]

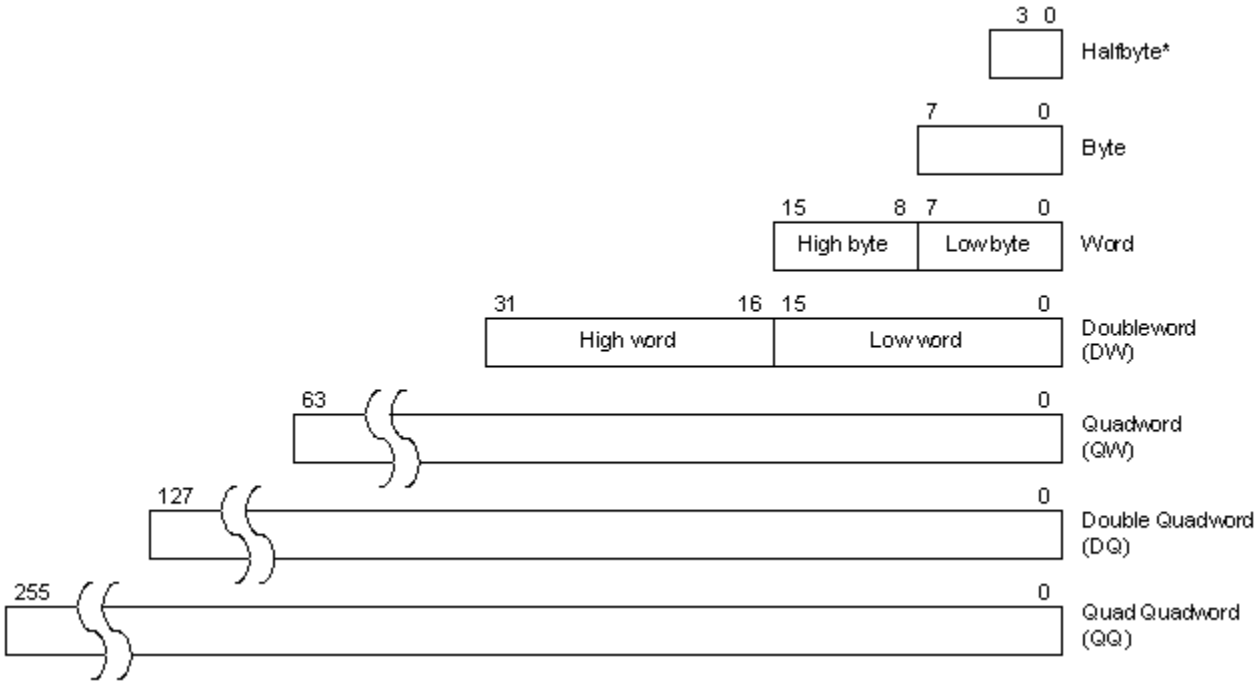
- IEEE Floating-Point Exceptions [CHV, BSW]
- Floating-Point Compare Operations [CHV, BSW]

Type Conversion

Fundamental Data Types

The fundamental data types in the GEN architecture are halfbyte, byte, word, doubleword (DW), quadword (QW), double quadword (DQ) and quad quadword (QQ). They are defined based on the number of bits of the data type, ranging from 4 bits to 256 bits. As shown in the figure below, a halfbyte contains 4 bits, a byte contains 8 bits, a word contains two bytes, a doubleword (DWord) contains two words, and so on. Halfbyte is a special data type that is not accessed directly as a standalone data element; it is only allowed as a subfield of the numeric data type of “packed signed halfbyte integer vector” described in the next section.

Fundamental Data Types



With the exception of halfbyte, the access of a data element to/from a GEN register or to/from memory must be aligned on the natural boundaries of the data type. The natural boundary for a word has an even-numbered address in units of bytes. The natural boundary for a doubleword has an address divisible by 4 bytes. Similarly, the natural boundary for a quadword, double quadword, and quad quadword has an address divisible by 8, 16, and 32 bytes, respectively. Double quadword, and quad quadword do not have corresponding numeric data types. Instead, they are used to describe a group (a vector) of numeric data elements of smaller size aligned to larger natural boundaries.

Numeric Data Types

The numeric data types defined in the GEN architecture include signed and unsigned integers and floating-point numbers (floats) of various sizes. These numeric data types are described below.

Integer Numeric Data Types

The Execution Unit supports the following integer data types. Signed integer types use two's complement representation for negative numbers.

UB: Unsigned Byte, 8-bit Unsigned Integer

7		0

B: Byte, 8-bit Signed Integer

7	6		0
S			

UW: Unsigned Word, 16-bit Unsigned Integer

15		0

W: Word, 16-bit Signed Integer

15	14		0
S			

UD: Unsigned Doubleword, 32-bit Unsigned Integer

31		0

D: Doubleword, 32-bit Signed Integer

31	30		0
S			

UQ: Unsigned Quadword, 64-bit Unsigned Integer CHV, BSW

63		0

Q: Quadword, 64-bit Signed Integer CHV, BSW

63	62		0
S			

UV: Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer

31		28	27		24	23		20	19		16	15		12	11		8	7		4	3		0

V: Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer

31		28	27		24	23		20	19		16	15		12	11		8	7		4	3		0
S			S			S			S			S			S			S			S		

The following table summarizes the EU integer data types.

Execution Unit Integer Data Types

Notation	Size in Bits	Name	Range
UB	8	Unsigned Byte Integer	[0, 255]
B	8	Signed Byte Integer	[-128, 127]
UW	16	Unsigned Word Integer	[0, 65535]
W	16	Signed Word Integer	[-32768, 32767]
UD	32	Unsigned Doubleword Integer	$[0, 2^{32} - 1]$
D	32	Signed Doubleword Integer	$[-2^{31}, 2^{31} - 1]$
UQ	64	Unsigned Quadword Integer	$[0, 2^{64} - 1]$
Q	64	Signed Quadword Integer	$[-2^{63}, 2^{63} - 1]$
UV	32	Packed Unsigned Half-Byte Integer Vector	[0, 15] in each of eight 4-bit immediate vector elements.
V	32	Packed Signed Half-Byte Integer Vector	[-8, 7] in each of eight 4-bit immediate vector elements.

Restriction: Only a raw move using the *mov* instruction supports a packed byte destination register region. For information about raw moves, refer to the **Description** in *mov – Move*.

Restriction: Q/UQ data types are not supported in Align16 mode.

Floating-Point Numeric Data Types

The Execution Unit supports the following floating-point data types.

Type or Description
The Float type uses the single precision format specified in IEEE Standard 754-1985 for Binary Floating-Point Arithmetic.
The Double Float type uses the double precision format specified in IEEE Standard 754-1985 for Binary Floating-Point Arithmetic.
In the ALT floating-point mode, representations for infinities, denorms, and NaNs within those formats are not used. The EU does not support the double extended precision (80-bit) floating-point format found in the x86/x87/Intel 64 floating-point registers. All floating-point formats are signed using signed magnitude representation (a distinct sign bit, separate from the magnitude information).
The Half Float type uses the binary16 format specified in IEEE Standard 754-2008.

Type or Description
The F (Float) type supports both the ALT and IEEE floating-point modes, controlled by the Single Precision Floating-Point Mode bit in the Control Register.
Whether IEEE mode F calculations support denorms or flush denormalized values to zero is controlled by the Single Precision Denorm Mode bit in the Control Register.
The DF (Double Float) type only supports the IEEE floating-point mode. Whether DF calculations support denorms or flush denormalized values to zero is controlled by the Double Precision Denorm Mode bit in the Control Register.
The HF (Half Float) type only supports the IEEE floating-point mode.
HF calculations flush denormalized values to zero and gradual underflow is not supported.

HF: Half Float, 16-bit Half-Precision Floating-Point Number CHV, BSW

15	14		10	9		0
S	biased exp.			fraction		

F: Float, 32-bit Single-Precision Floating-Point Number

31	30		23	22		0
S	biased exponent			fraction		

DF: Double Float, 64-bit Double-Precision Floating-Point Number CHV, BSW

63	62		52	51		0
S	biased exponent			fraction		

VF: Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number

31	30		28	27		24	23	22		20	19		16	15	14		12	11		8	7	6		4	3		0
S	b. exp.			frac.			S	b. exp.			frac.			S	b. exp.			frac.		S	b. exp.			frac.			

The following table summarizes the EU floating-point data types.

Execution Unit Floating-Point Data Types

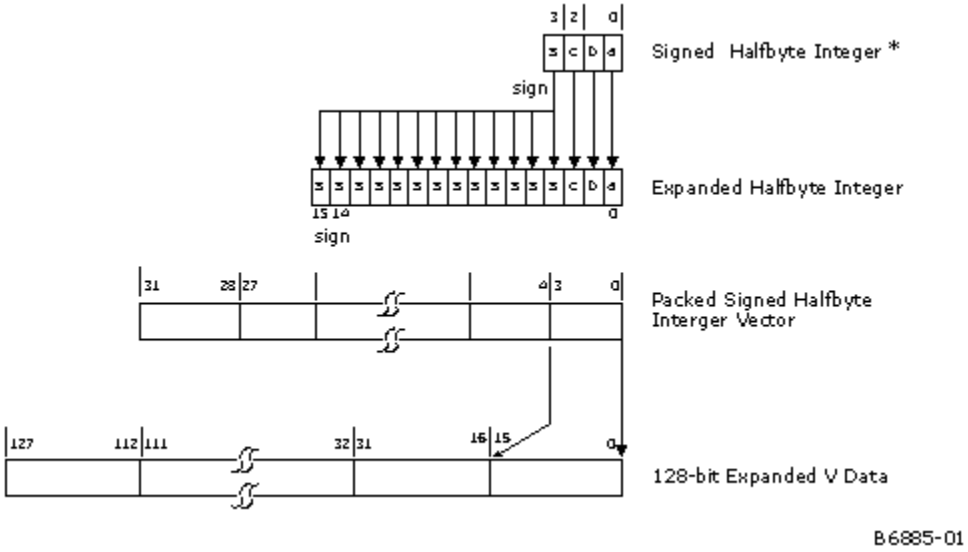
Notation	Size in Bits	Name	Range
HF	16	Half Float	Half precision, 1 sign bit, 5 bits for the biased exponent, and 10 bits for the significand: $[-(2-2^{-10})^{31}...-2^{-40}, 0.0, 2^{-40}... (2-2^{-10})^{31}]$
F	32	Float	Single precision, 1 sign bit, 8 bits for the biased exponent, and 23 bits for the significand: $[-(2-2^{-23})^{127}...-2^{-149}, 0.0, 2^{-149}... (2-2^{-23})^{127}]$
DF	64	Double Float	Double precision, 1 sign bit, 11 bits for the biased exponent, and 52 bits for the significand: $[-(2-2^{-52})^{1023}...-2^{-1074}, 0.0, 2^{-1074}... (2-2^{-52})^{1023}]$
VF	32	Packed Restricted Float Vector	Restricted precision. Each of four 8-bit immediate vector elements has 1 sign bit, 3 bits for the biased exponent (bias of 3), and 4 bits for the significand: [–31...–0.125, 0, 0.125... 31]

Packed Signed Half-Byte Integer Vector

A packed signed halfbyte integer vector consists of 8 signed halfbyte integers contained in a doubleword. Each signed halfbyte integer element has a range from -8 to 7 with the sign on bit 3. This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand or a non-immediate source operand. GEN hardware converts the vector into an 8-element signed word vector by sign extension. This is illustrated in *Numeric Data Types*.

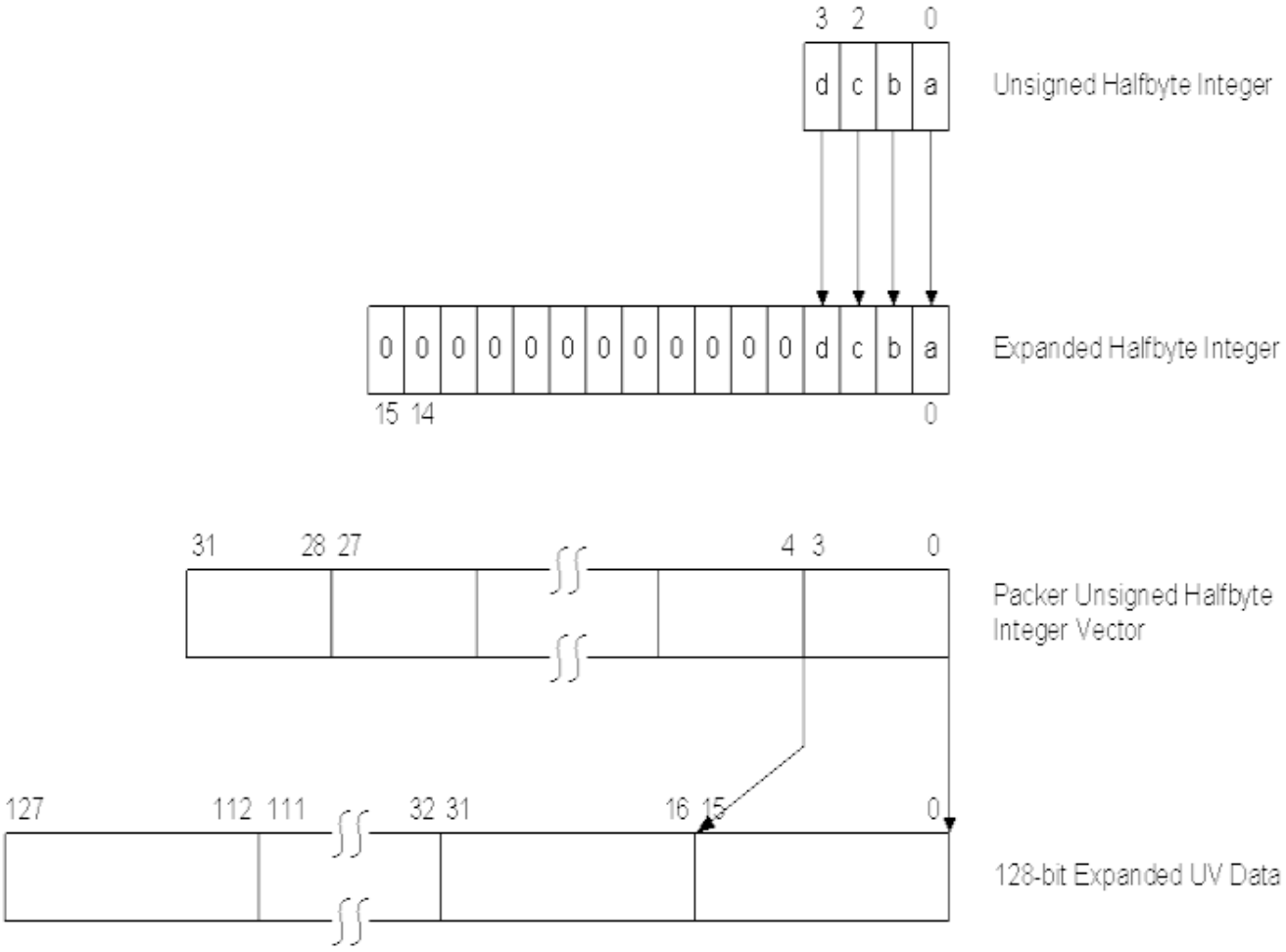
The short hand format notation for a packed signed half-byte vector is **V**.

Converting a Packed Half-Byte Vector to a 128-bit Signed Integer Vector



Packed UnSigned Half-Byte Integer Vector

A packed unsigned halfbyte integer vector consists of 8 unsigned halfbyte integers contained in a doubleword. Each unsigned halfbyte integer element has a range from 0 to 15. This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand or a non-immediate source operand. GEN hardware converts the vector into an 8-element signed word vector.

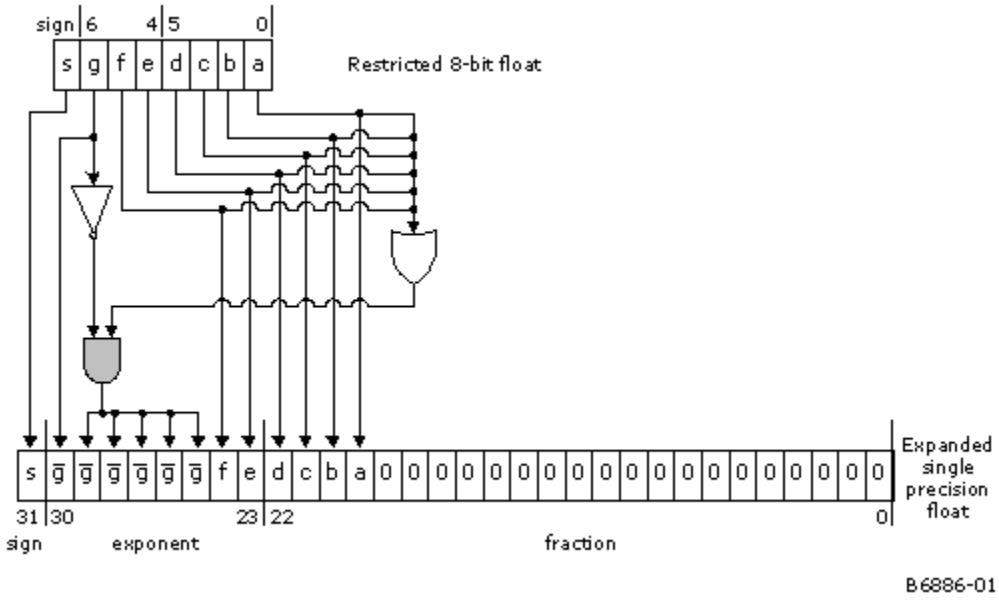


Packed Restricted Float Vector

A packed restricted float vector consists of 4 8-bit restricted floats contained in a doubleword. Each restricted float has the sign at bit 7, a 3-bit coded exponent in bits 4 to 6, a 4-bit fraction in bits 0 to 3, and an implied integer 1. The exponent is in excess-3 format – having a bias of 3. Restricted float provides zero, positive/negative normalized numbers with a small range (3-bit exponent) and small precision (4-bit fraction). This numeric data type is only used by an immediate source operand of doubleword in a GEN instruction. It cannot be used for the destination operand, or a non-immediate source operand.

The following figure shows how to convert an 8-bit restricted float into a single precision float. Converting a 3-bit exponent with a bias of 3 to an 8-bit exponent with a bias of 127 is by adding 4, or equivalently copying bit 2 to bit 7 and putting the inverted bit 2 to bits 6:2. A special logic is also needed to take care of positive/negative zeros.

Conversion from a Restricted 8-bit Float to a Single-Precision Float



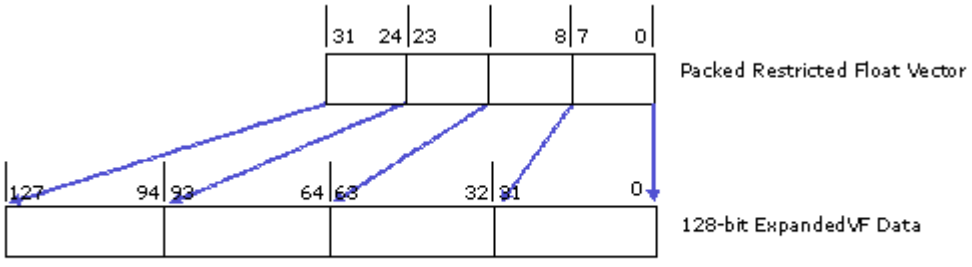
The following table shows all possible numbers of the restricted 8-bit float. Only normalized float numbers can be represented, including positive and negative zero, and positive and negative finite numbers. Normalized infinites, NaN, and denormalized float numbers cannot be represented by this type. It should be noted that this 8-bit floating point format does not follow IEEE-754 convention in describing numbers with small magnitudes. Specifically, when the exponent field is zero and the fraction field is not zero, an implied one is still present instead of taking a denormalized form (without an implied one). This results in a simple implementation but with a smaller dynamic range – the magnitude of the smallest non-zero number is 0.125.

Examples of Restricted 8-bit Float Numbers

Class	Hex #	Sign [7]	Exponent [6:4]	Fraction [3:0]	Extended 8-bit Exponent	Floating Number in Decimal
Positive Normalized Float	0x70-0x7F	0	111	0000 ... 1111	1000 0011	16 ... 31
	0x60-0x6F	0	110	0000 ... 1111	1000 0010	8 ... 15.5
	0x50-0x5F	0	101	0000 ... 1111	1000 0001	4 ... 7.75
	0x40-0x4F	0	100	0000 ... 1111	1000 0000	2 ... 3.875
	0x30-0x3F	0	011	0000 ... 1111	0111 1111	1 ... 1.9375
	0x20-0x2F	0	010	0000 ... 1111	0111 1110	0.5 ... 0.96875
	0x10-0x1F	0	001	0000 ... 1111	0111 1101	0.25 ... 0.484375
	0x01-0x0F	0	000	0001 ... 1111	0111 1100	0.125 ... 0.2421875
	0x00	0	000	0000	0000 0000	0 (+zero)
Negative Normalized Float	0xF0-0xFF	1	111	0000 ... 1111	1000 0011	-16 ... -31
	0xE0-0xEF	1	110	0000 ... 1111	1000 0010	-8 ... -15.5
	0xD0-0xDF	1	101	0000 ... 1111	1000 0001	-4 ... -7.75
	0xC0-0xCF	1	100	0000 ... 1111	1000 0000	-2 ... -3.875
	0xB0-0xBF	1	011	0000 ... 1111	0111 1111	-1 ... -1.9375
	0xA0-0xAF	1	010	0000 ... 1111	0111 1110	-0.5 ... -0.96875
	0x90-0x9F	1	001	0000 ... 1111	0111 1101	-0.25 ... -0.484375
	0x81-0x8F	1	000	0001 ... 1111	0111 1100	-0.125 ... -0.2421875
	0x80	1	000	0000	0000 0000	-0 (-zero)

The following figure shows the conversion of a packed exponent-only float to a 4-element vector of single precision floats.

The shorthand format notation for a packed signed half-byte vector is VF.



Floating Point Modes

GEN architecture supports two floating point operation modes, namely IEEE floating point mode (IEEE mode) and alternative floating point mode (ALT mode). Both modes follow mostly the requirements in IEEE-754 but with different deviations. The deviations will be described in details in later sections. The primary difference between these modes is on the handling of Infs, NaNs and denorms. The IEEE floating point mode may be used to support newer versions of 3D graphics API Shaders and the alternative floating point mode may be used to support early Shader versions. Taking DirectX 3D graphics API Shaders for example, shader models before version 3.0 may use the alternative floating point mode, while version 3.0 and following shader models may use the IEEE floating point mode.

These two modes are supported by all units that perform floating point computations, including GEN execution units, GEN shared functions like Extended Math, the Sampler and the Render Cache color calculator, and fixed functions like VF, Clipper, SF and WIZ. Host software sets floating point mode through the fixed function state descriptors for 3D pipeline and the interface descriptor for media pipeline. Therefore different modes may be associated with different threads running concurrently. Floating point mode control for EU and shared functions are based on the floating point mode field (bit 0) of *cr0* register.

IEEE Floating Point Mode

This topic is currently under development.

Partial Listing of Honored IEEE-754 Rules

Here is a summary of expected 32-bit floating point behaviors in GEN architecture. Refer to IEEE-754 for topics not mentioned.

- INF – INF = NaN
- 0 * (+/–)INF = NaN
- 1 / (+INF) = +0 and 1 / (–INF) = –0
    - (+/–)INF / (+/–)INF = NaN as A/B = A * (1/B)
- INV (+0) = RSQ (+0) = +INF, INV (–0) = RSQ (–0) = –INF, and SQRT (–0) = –0
- RSQ (–finite) = SQRT (–finite) = NaN
- LOG (+0) = LOG (–0) = –INF, LOG (–finite) = LOG (–INF) = NaN
- NaN (any OP) any-value = NaN with one exception for min/max mentioned below. Resulting NaN may have different bit pattern than the source NaN.
- Normal comparison with conditional modifier of EQ, GT, GE, LT, LE, when either or both operands is NaN, returns FALSE. Normal comparison of NE, when either or both operands is NaN, returns TRUE.
    - **Note:** Normal comparison is either a **cmp** instruction or an instruction with conditional modifier

- Special comparison **cmpn** with conditional modifier of EQ, GT, GE, LT, LE, when the second source operand is NaN, returns TRUE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns FALSE. **Cmpn** of NE, when the second source operand is NaN, returns FALSE, regardless of the first source operand, and when the second source operand is not NaN, but first one is, returns TRUE.
    - This is used to support the proposed IEEE-754R rule on **min** or **max** operations. For which, if only one operand is NaN, min and max operations return the other operand as the result.
- Both normal and special comparisons of any non-NaN value against +/– INF return exact result according to the conditional modifier. This is because that infinities are exact representation in the sense that +INF = +INF and –INF = –INF.
    - NaN is unordered in the sense that NaN != NaN.
- IEEE-754 requires floating point operations to produce a result that is the nearest representable value to an infinitely precise result, known as "round to nearest even" (RTNE). 32-bit floating point operations must produce a result that is within 0.5 Unit-Last-Place (0.5 ULP) of the infinitely precise result. This applies to addition, subtraction, and multiplication.
- All arithmetic floating point instructions does Round To Nearest Even at the end of the computation, except the round instructions.

Complete Listing of Deviations or Additional Requirements vs IEEE-754

For a result that cannot be represented precisely by the floating point format, the EU uses rounding to nearest or even to produce a result that is within 0.5 Unit-Last-Place(0.5 ULP) of the infinitely precise result.

Description
The rounding mode is specified by the Rounding Mode field in the Control Register.
Source modifiers are not applied to NAN.
Handle denorms as follows: • When inputs are denorms in mixed mode (one of the source operand is half float and other is single precision float OR source is half float and destination is single precision float), on upconversion, they are non-denorms. So computes in 32b can handle this. • When outputs are denorms, the denorms are retained if there is a format conversion. • Mixed mode operations should always behave like sum of individual operations.

Other information regarding floating-point behaviors:

- NaN input to an operation always produces NaN on output, however the exact bit pattern of the NaN is not required to stay the same (unless the operation is a raw “mov” instruction which does not alter data at all.)
- x*1.0f must always result in x (except denorm flushed and possible bit pattern change for NaN).

- x +/- 0.0f must always result in x (except denorm flushed and possible bit pattern change for NaN). But -0 + 0 = +0.
- Fused operations (such as *mac*, *dp4*, *dp3*, etc.) may produce intermediate results out of 32-bit float range, but whose final results would be within 32-bit float range if intermediate results were kept at greater precision. In this case, implementations are permitted to produce either the correct result, or else ±inf. Thus, compatibility between a fused operation, such as *mac*, with the unfused equivalent, *mul* followed by *add* in this case, is not guaranteed.
- As the accumulator registers have more precision than 32-bit float, any instruction with accumulator as a source/destination operand may produce a different result than that using more general registers, as indicated in this table:

Description
Such an instruction may produce a different result than that using GRF registers.

- API Shader divide operations are implemented as x*(1.0f/y). With the two-step method, x*(1.0f/y), the multiply and the divide each independently operate at the 32-bit floating point precision level (accuracy to 1 ULP).
- See the Type Conversion section for rules on converting to and from float representations.

Min Max of Floating Point Numbers

A special comparison called Compare-NaN is introduced in the GEN architecture to handle the difference of above mentioned floating-point comparison and the rules on supporting MIN/MAX. To compute the MIN or MAX of two floating-point numbers, if one of the numbers is NaN and the other is not, MIN or MAX of the two numbers returns the one that is not NaN. When two numbers are NaN, MIN or MAX of the two numbers returns source1.

When both the sources are NaN inputs, the special case in the section 2.4.8 Floating point Min/Max Operations describe the results.

When one source is SNAN, DX and IEEE treat the outputs differently. The special case section 2.4.8 Floating point Min/Max operations described the results.

Min and Max is supported by conditional select.

Note even though f0.0 is specified in the instruction, the flag register is not touched by this instruction.

The following tables detail the rules for this special compare-NaN operation for floating-point numbers. Notice that excepting “Not-Equal-To” comparison-NaN, last columns in all other tables have ‘T’.

Results of "Less-Than" Comparison-NaN – CMPN.L [CHV, BSW]

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	T
-Fin	F	T/F	T	T	T	T	T	T	T
-denorm	F	F	F	F	F	F	T	T	T
-0	F	F	F	F	T	F	T	T	T
+0	F	F	F	F	F	F	T	T	T
+denorm	F	F	F	F	F	F	T	T	T
+Fin	F	F	F	F	F	F	T/F	T	T
+inf	F	F	F	F	F	F	F	F	T
NaN	F	F	F	F	F	F	F	F	F

Results of "Greater-Than or Equal-To" Comparison-NaN – CMPN.GE

src0 src1	-inf	-Fin	-denorm	-0	+0	+denorm	+Fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	T
-Fin	T	T/F	F	F	F	F	F	F	T
-denorm	T	T	T	T	T	T	F	F	T
-0	T	T	T	T	F	T	F	F	T
+0	T	T	T	T	T	T	F	F	T
+denorm	T	T	T	T	T	T	F	F	T
+Fin	T	T	T	T	T	T	T/F	F	T
+inf	T	T	T	T	T	T	T	T	T
NaN	F	F	F	F	F	F	F	F	F

Alternative Floating Point Mode

The key characteristics of the alternative floating point mode is that NaN, Inf, and denorm are not expected for an application to pass into the graphics pipeline, and the graphics hardware must not generate NaN, Inf, or denorm as computation result. For example, a result that is larger than the maximum representable floating point number is expected to be flushed to the largest representable floating point number, i.e., +fmax. The fmax has an exponent of 0xFE and a mantissa of all one's, which is the same for IEEE floating point mode.

Note that this mode is applicable ONLY to Single Precision Float datatype.

Description
This also implies that ALT mode is not supported when Single precision datatype is involved in format conversion to double precision or half precision.
ALT_MODE is supported for Single Precision float ONLY. Hence, ALT_MODE is NOT supported in mixed mode operation.

Here is the complete list of the differences of legacy graphics mode from the relaxed IEEE-754 floating point mode.

- Any +/- INF result must be flushed to +/- fmax, instead of being output as +/- INF.
- Extended mathematics functions of log(), rsq(), and sqrt() take the absolute value of the sources before computation to avoid generating INF and NaN results.

Alternative Floating Point Mode shows the support of these differences in various hardware units.

Supported Legacy Float Mode and Impacted Units

IEEE-754 Deviations	VF	Clipper	SF	WIZ	EU	EM	Sampler	RC
Any +/- INF result flushed to +/- fmax	Y	Y	Y	Y	Y	Y	Y	Y
Log, rsq, sqrt take abs() of sources	N/A	N/A	N/A	N/A	N/A	Y	N/A	N/A

Alternative Floating Point Mode shows some of the desired or recommended alternative floating point mode behaviors that do not have hardware design impact. The reasons of not needing special hardware support for these items are also provided. This is based on the compliance requirement that can be found in the DirectX 9 specification: "**Handling of NaNs, Infs, and denorms is undefined. Applications should not pass in such values into the graphics pipeline.**"

Dismissed Legacy Behaviors

Suggested IEEE-754 Deviations	Reason for Dismiss
Mov forces (+/-)INF to (+/-)fmax	(+/-)INF is never present as input
(+/-)INF – (+/-)INF = +/- fmax instead of NaN	(+/-)INF is never present as input
Denorm must be flushed to zero in all cases (including trivial mov and point sampling)	Denorm is never present as input
Anything*0=0 (including NaN*0=0 and INF*0=0)	NaN and INF are never present as input
Except propagated NaN, NaN is never generated	NaN is never present as input and GEN never generates NaN based on rules in the previous table
An input NaN gets propagated excepting (a)-(d)	NaN is never present as input
(a) Rcp (and rsq) of 0 yields fmax	N/A, as it is already covered by the general rule “Any +/- INF result flushed to +/- fmax”
(b) Sampler honors 0/0 = 0 as if (1/0)*0	There is no divide in Sampler
I Rcp (and rsq) of INF yields +/- 0	(+/-)INF is never present as input
(d) Sampler honors INF/INF = 0 as if (1/INF)=0 followed by Anything*0 = 0	There is no divide in Sampler

Floating-Point Support

The following sections describe CHV, BSW floating-point support relative to the IEEE Standard for Floating-Point Arithmetic, currently IEEE Standard 754-2008. These sections cover binary floating-point arithmetic, as the EU provides no support for decimal floating-point arithmetic.

Note: Hardware alone is usually not fully conformant to the IEEE standard. It requires software functions to supplement the hardware.

Compared to previous generations, CHV, BSW does the following:

- Adds the HF (Half Float) type and a corresponding HF execution data type and execution path.
- Adds flags to indicate IEEE floating-point exceptions and to enable or disable exception reporting to those flags.
- Adds the Single Precision Denorm Mode bit in Control Register 0. It can be enabled to allow calculations using the F (Float) type in IEEE floating-point mode to support denormals and gradual underflow.

Floating-Point Types and Values

The EU supports 16-bit (HF, Half Float), 32-bit (F, Float), and 64-bit (DF, Double Float) types in the IEEE Standard formats (respectively *binary16*, *binary32*, and *binary64* in IEEE 754-2008). See Floating-Point Numeric Data Types for the layout of the supported floating-point types.

Any bit pattern for a floating-point value corresponds to a value defined by the standard: ± finite (normalized nonzero finite number), ± 0 (signed zero), ± inf (signed infinity), ± denorm (denormalized, very small but nonzero number), or NaN (Not a Number). A NaN can be a signaling NaN (sNaN) or quiet NaN (qNaN).

These operating modes are available for the different floating-point types:

- Half Float uses the IEEE floating-point mode.

Project	Description
CHV, BSW	Half Float does not support denormals. Flush denorms to zero when reading source operands and flush denorm calculation results to zero. Denorm flushing preserves sign.

- Float can use the ALT (Alternative Floating-Point Mode) or the IEEE floating-point mode. In IEEE mode, support for denormals and gradual underflow is controlled by the Single Precision Denorm Mode bit in the Control Register.
- Double Float uses the IEEE floating-point mode. Support for denormals and gradual underflow is controlled by the Double Precision Denorm Mode bit in the Control Register.

Flush to zero is not defined by IEEE Standard 754, but is implementation-specific and required by some APIs (including DirectX), thus the EU ISA supports either using or flushing Float or Double Float denorms based on the respective Denorm Mode bits.

Specifications in this volume sometimes reference ±fmax, the largest finite magnitude representable in a format, and ±fmin, the smallest normalized nonzero magnitude representable in a format. Calculating

those values uses the extreme exponent values for finite nonzero floating-point values, Emax and Emin below, along with the number of explicit fraction bits (not counting the implicit bit in the significand). The following table provides these values, with the fmax and fmin values generally approximate.

Floating-Point Type Parameters

Type	Exp. Bits	Exp. Bias	Emax	Emin	Explicit Fraction Bits	fmax	fmin
HF	5	15	15	-14	10	65504.0	about 6.1E-5
F	8	127	127	-126	23	about 3.4E38	about 1.18E-38
DF	11	1023	1023	-1022	52	about 1.79E308	about 2.23E-308

Where f is the number of explicit fraction bits, the general formula for fmax is $(2.0 - 2^{-f}) * 2^{Emax}$.

The general formula for fmin is 2^{Emin}.

Not a Number (NaN) Formats

A NaN has a biased exponent field with all bits set (as if encoding an exponent of Emax + 1), a nonzero fraction field (as a zero fraction field with that exponent indicates infinity), and either sign bit.

As specified in IEEE Standard 754-2008, the MSB of the fraction field, what would be the first bit following the binary point in a numeric value, determines a NaN's type:

0 - Signaling NaN (sNaN). The remaining fraction bits cannot all be zero.

1 - Quiet NaN (qNaN). The remaining fraction bits can have any value.

When an sNaN is an input, an operation normally signals the Invalid Operation exception and *quietizes* the NaN, producing the equivalent qNaN value, with MSB set to 1, at the output. Raw moves do not check for NaNs and do not signal exceptions or quietize NaN values.

When QNAN as one of the input to an operation this results in QNAN without raising the exception flag. This silently propagates and the output is the same QNAN as in the input.

Intel specifies the value *qNaN Indefinite* as a quiet NaN with all zeros in the remaining fraction bits, those other than the MSB. This value is useful because it is never produced by quietizing an sNaN, thus qNaN Indefinite may be used to initialize floating-point values that are not otherwise initialized by software, allowing the uninitialized case to be distinguished.

The EU applies numeric source modifiers (**-**, **(abs)**, or **-(abs)**) to NaN source values as well as to other values, possibly changing the sign bit of a NaN value when it is propagated. NaN sign bits are normally don't care values.

Per IEEE Standard 754-2008, a NaN's *payload* is contained in all fraction field bits other than the fraction MSB. Thus in the overall floating-point format, the sign bit, biased exponent, and fraction MSB are not part of the payload. NaN payload values are not affected by quietizing or by source modifiers. As noted above, an sNaN must have a nonzero payload and a qNaN can have any payload.

Floating-Point Rounding Modes

The EU supports the four rounding modes required in IEEE Standard 754 for binary floating-point arithmetic. If the unrounded result of infinite precision and range is exactly representable in the destination format, then that exact result is produced and no exception is signaled. For an unrounded result of infinite precision and range that is not exactly representable in the destination format (an *inexact* result), rounding chooses a numerically adjacent value in the destination format, while signaling the Inexact, Overflow, or Underflow exceptions when appropriate. The four rounding modes are:

RNE - Round to nearest or even. Choose the value in the destination precision nearest to the unrounded result. If the unrounded result is midway between two such values, choose the value with its least significant fraction bit as 0 (*even*).

RD - Round down, toward minus infinity.

RU - Round up, toward plus infinity.

RZ - Round toward zero.

The rounding mode is specified by the Rounding Mode field in the Control Register. It is initialized by Thread Dispatch. The normal default value is round to nearest or even. The Rounding Mode can be read to check the mode and written to change it. The Control Register and the Rounding Mode value are thread-specific; the Rounding Mode applies to all floating-point types, all execution channels, and all floating-point instructions executed by the thread after it is assigned.

Rounding an inexact result signals the Inexact Exception.

Rounding an inexact result preserves the sign of the result.

Infinities and NaNs are exact results and are not affected by the rounding mode.

Zeros are exact results, but the signs of zero results are affected by the rounding mode in certain cases:

X - X = +0 for RNE, RU, RZ

X - X = -0 for RD

(+0) + (-0) = (-0) + (+0) = +0 for RNE, RU, RZ

(+0) + (-0) = (-0) + (+0) = -0 for RD

Regardless of the rounding mode, (+0) + (+0) = +0 and (-0) + (-0) = -0.

The *directed rounding modes* are round down, round up, and round toward zero.

In IEEE mode, when a floating-point overflow occurs the result is determined by the sign of the result and the rounding mode:

+ and (RNE or RU): + inf

+ and (RD or RZ): + fmax

- and (RU or RZ): - fmax

- and (RNE or RD): - inf

Note that for floating-point overflow in IEEE mode, RNE always produces ± inf and RZ always produces ± fmax.

Floating-Point Operations and Precision

IEEE Standard 754-2008 requires the following floating-point operations to be precise within <= 0.5 *ulp* (unit in the last place) when using the round to nearest or even rounding mode:

ADD (*add* instruction)

DIV (*math* instruction with FDIV function code)

FMA (fused multiply add, *mad* instruction)

MUL (*mul* instruction)

SQRT (*math* instruction with SQRT function code)

SUB (*add* instruction using one **-** source modifier)

Conversions (float to float, float to int, and int to float)

Min/Max

Compare

Single Precision Floating-Point Rounding to Integral Values

The *rndd* (Round Down), *rnde* (Round to Nearest or Even), *rndu* (Round Up), and the *rndz* (Round to Zero) instructions round arbitary Float values to integral Float values. Each instruction specifies its rounding mode so these instructions are not affected by the Rounding Mode in the Control Register.

An integral source value produces the same value for the destination (ignoring any saturation). For magnitudes >= 8,388,608 (2^{23}) all Float values are integral.

The rounding instructions are sign preserving.

Signed zeros are propagated. In IEEE mode, signed infinities are propagated. In IEEE mode, sNaN inputs are quietized, the equivalent qNaN is produced, and the Invalid Operation exception is indicated. In IEEE mode, qNaN inputs are propagated.

No other exceptions are signaled for these instructions. For example if the source and result values differ, the Inexact exception is not signaled.

The Single Precision Denorm Mode in the Control Register affects the results of the *rndd* and *rndu* instructions for denorm source values.

Floating-Point to Integer Conversion

The *mov* and *sel* instructions can be used to convert floating-point values to integers. In the tables below, Imin is the smallest representable value in a signed integer type, Imax is the largest representable value in an integer type, and f is a finite floating-point value after rounding to an integral value using the current rounding mode.

Converting unrepresentable floating-point values, including infinities, NaNs, and values that convert to integers outside of the destination type's range, signal Invalid Operation exceptions. When the destination integer type is unsigned, normalized nonzero negative inputs signal Invalid Operation exceptions and negative denorm inputs signal Inexact exceptions.

Data written in accumulator using implicit or explicit destination will not be IEEE compliant.

Floating-Point to Integer Conversion Results and Exceptions for Signed Integer Types

FP Value	Integer Result	FP Exception
qNaN	0	Invalid Operation
sNaN	0	Invalid Operation
+inf	Imax	Invalid Operation
f > Imax	Imax	Invalid Operation
Imin <= f <= Imax	f	Inexact if rounding changed f
f < Imin	Imin	Invalid Operation
-inf	Imin	Invalid Operation

Floating-Point to Integer Conversion Results and Exceptions for Unsigned Integer Types

FP Value	Integer Result	FP Exception
qNaN	0	Invalid Operation
sNaN	0	Invalid Operation
+inf	Imax	Invalid Operation
f > Imax	Imax	Invalid Operation
0 <= f <= Imax	f	Inexact if rounding changed f
f = -0	0	Inexact
-1 < f < -0	0	Inexact
-1 < f < -0	Real Indefinite	Invalid Operation
-fmax <= f <= -1	Real Indefinite	Invalid Operation
-inf	Real Indefinite	Invalid Operation

Note: Real Indefinite is encoded as Integer value 0.

Integer to Floating-Point Conversion

Integer to floating-point conversion follows these rules in IEEE mode:

- The result is never ± inf, never NaN, and never -0.
- If the integer source value is not exactly representable in the destination floating-point format, use the current rounding mode to choose an adjacent floating-point value and signal the Inexact Exception.
- If the integer source value is too large to represent in the destination floating-point format (only possible when converting to Half Float from D, UD, or UW) then signal the Overflow Exception. Based on the sign and the current rounding mode, the result is ± fmax or ± inf, as described in the Overflow Exception section.

Floating-Point Min/Max Operations

In the following Min/Max operations, sNaN inputs are preferred to non-NaN inputs and non-NaN inputs are preferred to qNaN inputs.

Min(x, qNaN) = Min(qNaN, x) = x with no exceptions signaled.

Min(x, sNaN) = Min(sNaN, x) = qNaN (quietized value corresponding to the input sNaN) and signal the Invalid Operation exception.

Note: DX deviates from this rule:

The DX behavior is inferred from the denorm bit.

Min(x, sNaN) = Min(sNaN, x) = x

Max(x, qNaN) = Max(qNaN, x) = x with no exceptions signaled.

Max(x, sNaN) = Max(sNaN, x) = qNaN (quietized value corresponding to the input sNaN) and signal the Invalid Operation exception.

Note: DX deviates from this rule:

The DX behavior is inferred from the denorm bit.

Max(x, sNaN) = Max(sNaN, x) = x

Special cases(when both sources are NaN inputs)

Min(qNaN, qNan) = qNaN (of the first source) and no exception raised

Min(qNaN, sNaN) = Min(sNaN, qNaN) = qNaN (quiet-ized sNaN and signal the Invalid Operation exception)

Min(sNaN, sNaN) = qNaN(of the first source and signal Invalid Operation Exception)

Max(qNaN, qNan) = qNaN (of the first source) and no exception raised

Max(qNaN, sNaN) = Max(sNaN, qNaN) = qNaN (quiet-ized sNaN and signal the Invalid Operation exception)

Max(sNaN, sNaN) = qNaN(of the first source and signal Invalid Operation Exception)

IEEE Floating-Point Exceptions

The EU detects the five floating-point exceptions defined by IEEE Standard 754:

Invalid Operation

Division by Zero

Overflow

Underflow

Inexact

Description
When converting from float to int, IEEE exception flags are not updated correctly.
Non IEEE compliant instructions updating exception flags
float_to_int conversion is supported in HW only for RTZ mode

Signaling Floating-Point Exceptions

When enabled in the Control Register, floating-point exceptions are detected and set *sticky* flag bits in the State Register. There is no mechanism for automatic transfer to a handler, so floating-point exceptions are not handled like other exceptions described in the Exceptions chapter.

Setting flags to indicating Floating-Point exceptions is the default exception handling approach specified by IEEE Standard 754. The flag bits are sticky because in normal operation the EU only sets these bits as exceptions occur, and does not clear these bits, so a set value sticks until cleared by software. The Control Register and State Register are cleared at reset and initialized at thread load. Both are read/write registers. These fields are used:

- **IEEE Exception Trap Enable** (Control Register cr0.0:ud bit 9). This bit enables trapping IEEE exception flags. This control bit may be updated by software. It is initially zero on thread load. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exception field of sr0.1, in the State Register. **Note:** Software must set this flag at thread start to use the IEEE Exception flags.
- **IEEE Exception**. (State Register sr0.1:ud bits 7:0). The IEEE exception bits are sticky bits set by the opcodes when floating-point exceptions are triggered. These bits are defined per thread and all channels update one set of sticky bits. These bits are cleared on thread load and may be cleared by software. Exception updates to these bits may be disabled by clearing the IEEE Exception Trap Enable bit in the Control Register. The following table specifies the IEEE exception bits:

Bits	Definition
7:5	Reserved
4	Inexact Exception
3	Overflow Exception
2	Underflow Exception

Bits	Definition
1	Division by Zero Exception
0	Invalid Operation Exception

The IEEE exception flags are per thread, shared by all channels. (Maintaining separate per channel exception flags for 32 channels would require 160 bits per thread.)

Invalid Operation Exception

An Invalid Operation exception is signaled by any operation on a signaling NaN (sNaN) or by certain combinations of operations and operands with undefined results, always producing a quiet NaN (qNaN) result.

The following specific operations signal Invalid Operation, where *x* is a positive, finite, nonzero, and normalized number:

+inf - (+inf) or (-inf) - (-inf)

± 0 / ± 0

± inf / ± inf

± 0 × ± inf or ± inf × ± 0

Remainder(± *x*, ± 0)

Remainder(± inf, ± *x*)

Sqrt(-*x*)

Note that Sqrt(-0) is -0 per IEEE Standard 754 and does not cause any exception.

These instructions can signal specific Invalid Operation exceptions (and also on sNaN inputs except for Float inputs in ALT mode), producing a qNaN result:

add

dp2, *dp3*, *dp4*, and *dph* (the Dot Product instructions)

line

lrp

mac

mad

madm

math with the FDIV, SQRT, or RSQRTM function codes

mul

pln

These other instructions signal Invalid Operation exceptions on sNaN inputs (except for Float inputs in ALT mode), producing a qNaN result:

cmp, *cmpn*

frc

math with all other function codes for floating-point operations

mov except for raw moves

rndd, *rnde*, *rndu*, and *rndz* (the Round instructions)

sel

smov except for raw moves

Division by Zero Exception

The operation ± *x* / ± 0, where *x* is a positive, finite, nonzero, and normalized number, signals the Division by Zero Exception and produces a correctly signed ± inf result. Note that in accordance with the standard, ± 0 / ± 0 signals Invalid Operation, does not signal Division by Zero, and produces a qNaN result. This behavior can occur for the *math* instruction with the FDIV function code.

The operation $LOG_2(\pm 0)$ signals the Division by Zero Exception and produces -inf as the result. This behavior can occur for the *math* instruction with the LOG function code.

Overflow Exception

A floating-point *overflow* occurs when an operation with a finite result produces an internal result with magnitude > fmax, the maximum representable finite value in the destination format. The internal result is rounded to the destination precision with the current rounding mode but has an unbounded exponent. An overflow produces ± inf or ± fmax as the result, depending on the sign and the rounding mode.

The following algorithm describes floating-point overflow processing:

1. Compute the result with infinite precision and unbounded range.
2. Normalize the result using an unbounded exponent.
3. Round the result to the destination precision using an unbounded exponent and the current rounding mode.
4. If (abs(rounded unbounded result) > fmax(destination format)) {

   Set the Overflow Exception flag to 1.

   Output = ± inf or ± fmax depending on the sign and the rounding mode:

   + and (RNE or RU): + inf

   + and (RD or RZ): + fmax

   - and (RU or RZ): - fmax

- and (RNE or RD): - inf

}

}

Note: The first three steps of the overflow and underflow algorithms are identical.

Underflow Exception

An *underflow* occurs when an operation produces a tiny but nonzero inexact result *x*, with abs(*x*) < fmin, where fmin = 2^{Emin}. See the Floating-Point Type Parameters table for the fmin and Emin values for different floating-point types.

IEEE Standard 754 allows underflow to be determined before or after rounding. The Execution Unit determines underflow after rounding, which is consistent with the behavior of the x87 and SSE instructions in the CPU.

When denorms are flushed to zero, no underflow exceptions are signaled. Flush to zero is not defined by IEEE Standard 754, but is implementation specific and required by some APIs (including DirectX), thus the EU ISA supports either using or flushing Float or Double Float denorms based on the respective Denorm Mode bits.

When denorms are enabled, if an operation's internal result rounded to the destination precision, but using an unbounded exponent range, has a magnitude that is less than fmin but nonzero AND the denorm result in the destination format is inexact, then signal the Underflow Exception.

The rounded result can be ± 0, ± fmin, or (when denorms are enabled) ± denorm.

IEEE Standard 754 requires the non-intuitive behavior that an exact denorm result does not set the Underflow Exception flag.

The following algorithm describes floating-point underflow processing:

1. Compute the result with infinite precision and unbounded range.
2. Normalize the result using an unbounded exponent.
3. Round the result to the destination precision using an unbounded exponent and the current rounding mode.
4. If (0 < abs(rounded unbounded result) < fmin(destination format) {

   if (flush denorms to zero) {

   Output = 0; // No underflow exception.

   }

   Else {

   Renormalize to the bounded exponent with the original infinite precision value...

   ...and round that value to the destination precision using the current rounding mode.

If (the just computed value differs from the value computed in step (3) in exponent or mantissa)
{

Set the Underflow Exception flag to 1.

} // Note: Underflow is not set if the tiny result is the same as when computed with an unbounded exponent.

Output = rounded result using the destination precision and destination exponent range;

}

}

Else { // Not a tiny number.

Output = rounded number;

}

Inexact Exception

An Inexact Exception occurs when the internal unrounded result, with infinite precision and unbounded exponent range, differs from the generated result after format conversion, normalizing or denormalizing, and rounding. An Inexact Exception occurs irrespective of any saturation to exact zero or exact +1.0. The Inexact Exception is normal and may occur more often than not. For example, the calculation 1.0 / 3.0 is inexact in any binary floating-point format. These rules determine whether a result is inexact:

- Infinities and NaNs are never inexact.
- Flushing a denorm internal result to zero (always for Half Float and if the appropriate Denorm Mode is 0 for Float and Double Float) is always inexact.
- If any rounding occurs using the current Rounding Mode, so the rounded result differs from the internal unrounded result, the result is inexact. However explicit round to integral using any of the rounding instructions (*rndd*, *rnde*, *rndu*, and *rndz*) is never inexact.

Floating-Point Compare Operations

Four mutually exclusive relations are possible between two floating-point values, src0 and src1:

Less than. src0 < src1 and neither source is NaN.

Equal. src0 = src1 and neither source is NaN.

Greater than. src0 > src1 and neither source is NaN.

Unordered. Any source is NaN.

Any NaN compares unordered to any value, including itself.

Infinities of the same sign compare as equal.

Zeros compare as equal regardless of sign: -0 = +0.

Floating-Point Compare Relations

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	E	L	L	L	L	L	L	L	U
-fin	G	*	L	L	L	L	L	L	U
-denorm	G	G	*^	L^	L^	L^	L	L	U
-0	G	G	G^	E	E	L^	L	L	U
+0	G	G	G^	E	E	L^	L	L	U
+denorm	G	G	G^	G^	G^	*^	L	L	U
+fin	G	G	G	G	G	G	*	L	U
+inf	G	G	G	G	G	G	G	E	U
NaN	U	U	U	U	U	U	U	U	U

Notes	
*	Relation can be L, E, or G.
^	When denorms are flushed to zero then all denorms and zeros compare as E.

The next six tables show the results of six specific comparisons, corresponding to the **.g**, **.l**, **.e**, **.ne**, **.ge**, and **.le** conditional modifiers used with the *cmp* instruction and a floating-point source type. Any NaN source produces a false comparison result for these modifiers other than **.ne** and produces a true comparison result for the **.ne** modifier.

Results of Greater Than Comparison — *cmp*.g

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	F	F	F^	F	F	F
+0	T	T	T^	F	F	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes

src0 \ src1	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	F	F	F^	F	F	F
+0	T	T	T^	F	F	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.g** comparison result F.

Results of Less Than Comparison — *cmp*.l

src0 \ src1	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	F
-fin	F	*	T	T	T	T	T	T	F
-denorm	F	F	*^	T^	T^	T^	T	T	F
-0	F	F	F^	F	F	T^	T	T	F
+0	F	F	F^	F	F	T^	T	T	F
+denorm	F	F	F^	F^	F^	*^	T	T	F
+fin	F	F	F	F	F	F	*	T	F
+inf	F	F	F	F	F	F	F	F	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.l** comparison result F.

Results of Equal Comparison — *cmp*.e

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	F
-fin	F	*	F	F	F	F	F	F	F
-denorm	F	F	*^	F^	F^	F^	F	F	F
-0	F	F	F^	T	T	F^	F	F	F
+0	F	F	F^	T	T	F^	F	F	F
+denorm	F	F	F^	F^	F^	*^	F	F	F
+fin	F	F	F	F	F	F	*	F	F
+inf	F	F	F	F	F	F	F	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.e** comparison result T.

Results of Not Equal Comparison — *cmp*.ne

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	F	T	T	T	T	T	T	T	T
-fin	T	*	T	T	T	T	T	T	T
-denorm	T	T	*^	T^	T^	T^	T	T	T
-0	T	T	T^	F	F	T^	T	T	T
+0	T	T	T^	F	F	T^	T	T	T
+denorm	T	T	T^	T^	T^	*^	T	T	T
+fin	T	T	T	T	T	T	*	T	T
+inf	T	T	T	T	T	T	T	F	T
NaN	T	T	T	T	T	T	T	T	T

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.ne** comparison result F.

Results of Greater Than or Equal Comparison — *cmp*.ge

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	F	F	F	F	F	F	F	F
-fin	T	*	F	F	F	F	F	F	F
-denorm	T	T	*^	F^	F^	F^	F	F	F
-0	T	T	T^	T	T	F^	F	F	F
+0	T	T	T^	T	T	F^	F	F	F
+denorm	T	T	T^	T^	T^	*^	F	F	F
+fin	T	T	T	T	T	T	*	F	F
+inf	T	T	T	T	T	T	T	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.ge** comparison result T.

Results of Less Than or Equal Comparison — *cmp*.le

src1 / src0	-inf	-fin	-denorm	-0	+0	+denorm	+fin	+inf	NaN
-inf	T	T	T	T	T	T	T	T	F
-fin	F	*	T	T	T	T	T	T	F
-denorm	F	F	*^	T^	T^	T^	T	T	F
-0	F	F	F^	T	T	T^	T	T	F
+0	F	F	F^	T	T	T^	T	T	F
+denorm	F	F	F^	F^	F^	*^	T	T	F
+fin	F	F	F	F	F	F	*	T	F
+inf	F	F	F	F	F	F	F	T	F
NaN	F	F	F	F	F	F	F	F	F

Notes	
*	Result can be T or F.
^	When denorms are flushed to zero then all denorms and zeros compare as equal, making the **.le** comparison result T.

Type Conversion

This topic is currently under development.

Float to Integer

Converting from float to integer is based on rounding toward zero(RTZ is for DX, IEEE expects all four rounding modes). If the floating point value is +0, -0, +Denorm, -Denorm, +NaN –r -NaN, the resulting integer value is always 0. If the floating point value is positive infinity (or negative infinity), the conversion result takes the largest (or the smallest) represent-able integer value. If the floating point value is larger (or smaller) than the largest (or the smallest) represent-able integer value, the conversion result takes the largest (or the smallest) represent-able integer value. The following table shows these special cases. The last two rows are just examples. They can be any number outside the represent-able range of the output integer type (UD, D, UW, W, UB and B).

Input Format	Output Format					
F	UD	D	UW	W	UB	B
+/- Zero	00000000	00000000	00000000	00000000	00000000	00000000
+/- Denorm	00000000	00000000	00000000	00000000	00000000	00000000
NAN	00000000	00000000	00000000	00000000	00000000	00000000
-NAN	00000000	00000000	00000000	00000000	00000000	00000000
INF	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F
-INF	00000000	80000000	00000000	00008000	00000000	00000080
$+2^{32}$ (*)	FFFFFFFF	7FFFFFFF	0000FFFF	00007FFF	000000FF	0000007F
$-2^{32}-1$ (*)	00000000	80000000	00000000	00008000	00000000	00000080

Integer to Integer with Same or Higher Precision

Converting an unsigned integer to a signed or an unsigned integer with higher precision is based on zero extension.

Converting an unsigned integer to a signed integer with the same precision is based on modular wrap-around. Without saturation, a larger than represent-able number becomes a negative number. With saturation, a larger than represent-able number is saturated to the largest positive represent-able number.

Converting a signed integer to a signed integer with higher precision is based on sign extension.

Converting a signed integer to an unsigned integer with higher precision is based on sign extension. Without saturation, a negative number becomes a large positive number with the sign bit wrapped-up. With saturation, a negative number is saturated to zero.

Integer to Integer with Lower Precision

Converting a signed or an unsigned integer to a signed or an unsigned integer with lower precision is based on bit truncation. Without saturation, only the lower bits are kept in the output regardless of the sign-ness of input and output. With saturation, a number that is outside the represent-able range is saturated to the closest represent-able value.

Integer to Float

Converting a signed or an unsigned integer to a single precision float number is to round to the closest representable float number. For any integer number with magnitude less than or equal to 24 bits, resulting float number is a precise representation of the input. However, if it is more than 24 bits, by default a “round to nearest even” is performed.

Double Precision Float to Single Precision Float [CHV, BSW]

.

Double Precision Float	Single Precision Float
-inf	-inf
-finite	-finite/-denorm/-0
-denorm	-0
-0	-0
+0	+0
+denorm	+0
+finite	+finite/+denorm/+0
+inf	+inf
NaN	NaN

The upper Dword of every Qword will be written with undefined value when converting DF to F.

Single Precision Float to Double Precision Float [CHV, BSW]

Converting a single precision floating-point number to a double precision floating-point number will produce a precise representation of the input.

Single Precision Float	Double Precision Float
-inf	-inf
-finite	-finite
-denorm	-finite
-0	-0
+0	+0
+denorm	+finite
+finite	+finite
+inf	+inf
NaN	NaN

Exceptions

The GEN Architecture defines a basic exception handling mechanism for several exception cases. This mechanism supports both normal operations such as extensions of the mask-stack depth, as well as detecting some illegal conditions.

Exception Types

Type	Trigger / Source	Sync/Async Recognition
Software Exception	Thread code	Synchronous
Breakpoint	• A bit in the instruction word • Breakpoint IP match • Breakpoint Opcode match	Synchronous
Illegal Opcode	Hardware	Synchronous
Halt	MMIO register write	Asynchronous
Context Save/Restore	Preemption Interrupt	Asynchronous

Threads may choose which exceptions to recognize and which to ignore. This mask information is specified on a per-kernel basis in fixed function state generated by the driver, and delivered to the EU as part of a new thread dispatch. Upon arrival at the EU, the exception-mask information is used to initialize the exception enable fields of that thread's cr0.1 register, which controls exception recognition. This register is instantiated on a per-thread basis, allowing independent control of exception type recognition across hardware threads. The exception enable bits in the cr0.1 register are read/write, and thus can be enabled/disabled via software at any time during thread execution.

The exception handling mechanism relies on the System Routine, a single subroutine that provides common exception handling for all threads on all EUs in the system. This System Routine is defined per-context and is identified via a System IP (SIP) register in context state. At the time of each context switch, the appropriate SIP for that context is loaded into each EU, allowing each context to have custom implementation of exception handling routines if so desired.

The mechanism does not support handling recursive system routine access. This means when a thread cannot be asynchronously interrupted to an exception when executing a SIP.

Example:

An Exception is not supported when hardware is executing a SIP for context save and restore operations.

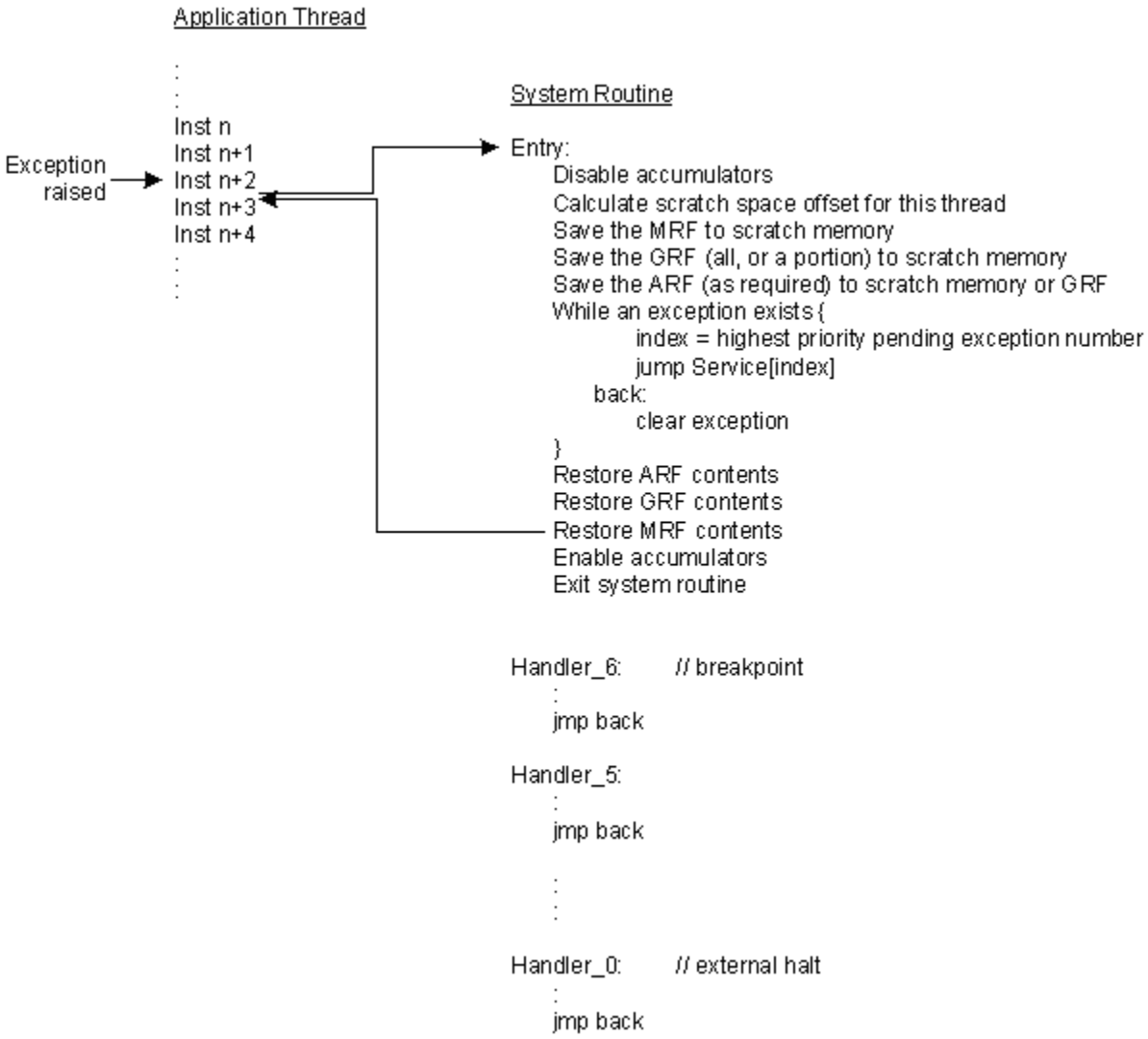
Exception-Related Architecture Registers

Exception-related registers are architecture registers cr0.0 through cr0.2. These registers are instantiated on a per-thread basis providing each hardware thread with unique control over exception recognition and handling. The registers provide the capability to mask exception types, determine the type of raised exception, store the return address, and control exiting from the System Routine back to the application thread.

Many of the bits in these registers are manipulated by both hardware and software. In all cases, the read/write operations by hardware and software occur at exclusive times in a thread's lifetime, thus there is no need for atomic read-modify-write operations when accessing these registers.

System Routine

The following diagram illustrates the basic flow of exception handling and the structure of the System Routine.



Invoking the System Routine

The System Routine is invoked in response to a raised exception. Once an exception is raised, no further instructions from the application thread are issued until the System Routine has executed and returned control back to the application thread.

After an exception is recognized by hardware, the EU saves the thread's IP into its AIP register (cr0.2), and then moves the System Routine offset, SIP, into the thread's IP register. At this point the next instruction issued for that thread is the first instruction of the System Routine.

The System Routine maintains the same execution priority, GRF register space, and thread state as the application thread from which it is invoked.

Due to assuming the same priority, there may be significant absolute time between an exception being raised and invoking the System Routine, as other higher priority threads within the EU continue to execute. From a thread's perspective, once an exception is recognized, the next instruction issued is from the System Routine.

At the time of System Routine invocation, there may still be outstanding registers in-flight from the application thread. Depending on the instruction sequence in the System Routine, an in-flight register may be referenced by the System Routine and cause a register-in-flight dependency. These dependencies are honored by the System Routine and may cause the System Routine to be suspended until the register retires.

Exception processing is not nested within the System Routine. If a future exception is detected while executing the System Routine, the exception is latched into cr0.1, but does not cause a nested re-invocation of the System Routine. The exception recognition hardware recognizes only one outstanding exception of each type; i.e., once a specific exception type is detected and latched in cr0.1, and until the exception is cleared, any further exception of that type is lost.

Accumulators are not natively preserved across the System Routine. To make sure the accumulators are in the identical state once control is returned to the application thread, the System Routine must either set the Accumulator Disable bit of cr0.0 before using any instruction that modifies an accumulator, or save and restore the accumulators (using GRF registers or system thread scratch memory) around the System Routine. Saving and restoring accumulators, including their extended precision bits, can be accomplished by a short series of moves and shifts of the accumulator register. Also note that the state of the Accumulator Disable bit itself must be preserved unless, by convention, the driver software limits its manipulation to only the System Routine.

Further, upon System Routine entry, the execution-related masks (Continue, Loop, If, and Active masks, contained in the Mask Register) will remain set as they were in the application thread. Thus only a subset of channels may be active for execution. To enable execution on all channels, the System Routine may choose to use the instruction option ‘NoMask’, or may choose to set the mask registers to the desired value so long as it saves/restores the original masks upon System Routine entry/exit.

Similarly there is no hardware mechanism to preserve flags, mask-stacks, or other architecture registers across the System Routine. The System Routine must ensure that these values are preserved (see the *Conditional Instructions Within the System Routine*section for related information).

Returning to the Application Thread

Prior to returning control to the application thread, the System Routine should clear the proper Exception Status and Control bit in cr0.1. Failure to do so forces the thread's execution to reenter the System Routine before any further instructions are executed from the application thread. (Note that single-stepping functionality is the one exception where the exception's Status and Control bit is not reset before exit.)

The System Routine may choose to loop within a single invocation of the System Routine until all pending exceptions are serviced, or may choose to service exceptions one at a time (a simpler solution, but less efficient).

The System Routine is exited, and control returned to the application thread, via a write to the Master Exception State and Control bit in cr0.0. Upon clearing this bit, the value of AIP (cr0.2) is restored to the thread's IP register and, with no further exceptions pending, execution resumes at that address. The System Routine must follow any write to the Master Exception State and Control bit with at least one SIMD-16 *nop* instruction to allow control to transition. Throughout the System Routine, the AIP register maintains its value at the time the exception was raised unless directly modified by the System Routine. (See the AIP register definition for specifics on the IP value saved to AIP).

System IP (SIP)

The System IP (SIP) is the 16 byte-aligned offset of the first instruction of the System Routine, relative to the General State Base Address. SIP is assigned by the STATE_SIP command to the command streamer which updates SIP in the EU.

The SIP is widened to 48 bits. However the EU still only uses the low 32 bits (bits 31:4 with bits 3:0 as zero).

When the System Routine is invoked, the application thread's current IP is first saved into the AIP field of cr0.2. The SIP address is then loaded into the thread's IP register and execution continues within the System Routine. Thus each invocation of the System Routine has a common entry point. Returning from the System Routine loads IP from AIP, continuing thread execution.

The SIP used for GPGPU Context Save routine must follow these rules:

a. All FC and MSG registers must write to odd GRF register and all other registers must write to even GRF register. This may change the context image and affects the context restore sequence accordingly.

b. IEEE macro operations must never be used if preemption is desired.

System Routine Register Space

The System Routine uses the same GRF space as the thread that invokes it.

As such all of the calling thread's registers and their contents are visible to the System Routine. Further, the System Routine must only use r0..r15 of the GRF, as a minimal thread may have requested and been allocated this few. If the System Routine requires more registers than this, the driver should establish a higher minimum allocation for all threads.

The System Routine may encounter any residual register dependencies of the calling thread until such time that they clear by the return of in-flight writebacks.

Only one 32-bit GRF location, r0.4, is reserved for System Routine use. This location is sufficient to allow the System Routine to calculate the appropriate offset of its private scratch memory in the larger system scratch memory space (as dictated by binding table entry 254). The offset is left as a driver convention, but is likely based on a combination of Thread and EU IDs (see the example system handler in the) section. Other than the reserved r0.4 register field, there is no explicit GRF register space dedicated to the System Routine, and any GRF needs must be accomplished via (a) convention between the System

Routine and application code, or (b) the System Routine temporarily spilling the thread's GRF register contents to scratch memory and restoring those contents before System Routine exit.

No persistent storage is automatically allocated to the System Routine, although a driver implementation may set aside part of system scratch memory for the System Routine.

Any parameter passing to the System Routine (for use by software exceptions) is done via the GRF based on system thread/application thread convention.

System Scratch Memory Space

There is a single unified system scratch memory space per context shared by all EUs. It is anticipated that block is further partitioned into a unique scratch sub-space per thread via conventions implemented in the System Routine, with each hardware thread having a uniform block size at a calculated offset from the base address. The block address for a thread can be based on an offset derived from the thread's execution unit ID and thread ID made available through the TID and EUID field of architecture register sr0.0.

Per_Thread_Block_Size = System_Scratch_Block_Size / (EU_Count * Thread_Per_EU);

Offset = (sr0.0.EID * Threads_Per_EU + sr0.0.TID) * Per_Thread_Block_Size;

where in GEN:

Threads_Per_EU = 4

EU_Count = 8

System_Scratch_Block_Size is a driver choice.

Access to system scratch memory is performed through the Data Port via linear single register or block-based read/write messages. The driver may choose to use any binding table index for system scratch surface description. As a practical matter, the same index is expected to be used across all binding tables, as the index is typically hard coded in Data Port messages used within the System Routine coupled with the fact that a single System Routine is used for all threads. Read/write messages to the Data Port contain the address of the binding table (provided in r0 of all threads) and an offset, from which the Data Port calculates the final target address.

It is expected that the system scratch memory space is allocated by the driver at context-create time and remains persistent at a constant memory address throughout a context's lifetime.

Conditional Instructions Within the System Routine

It is expected that most, if not all, control flow within the System Routine is scalar in nature. If so, the System Routine should set SPF (Single Program Flow, cr0.0) to enable scalar branching. In this mode, conditional/loop instructions do not update the mask stacks and therefore do not have restrictions on their use nor require the save/restore of hardware mask stack registers.

If SIMD branching is desired within the System Routine, special considerations must be taken. Upon entry to the System Routine, the depth of the mask stacks is unknown at that point, and may be near full. If so, a subsequent conditional instruction and its associated mask ‘push’ may cause a stack

overflow. This would generate an exception within the system routine, an unsupported occurrence. To prevent this, if the System Routine uses SIMD conditional instructions, it must save the mask stacks prior to the first SIMD conditional instruction, and restore them after the last SIMD conditional instruction. As a general solution, it may be easiest to simply implement the save/restore as part of the entry/exit code sequence, using an available GRF register pair as a storage location. Once saved, the stacks should be reset to their empty condition, namely depth = 0 and top of stack = 0xFFFFFFFF.

Use of NoDDClr

The GEN instruction word defines an instruction option **NoDDClr** that overrides the native register dependency clearing mechanism of the typical instruction. When specified, NoDDClr does not clear, at register writeback time, the dependency placed on the destination register of the instruction. Use of this mechanism may provided increased performance when a kernel can guarantee no dependency issues between instructions, but may cause issues with exception handling in some circumstances as discussed here.

Typically NoDDClr is used in an instruction series to enable a sequence of writes to sub-fields of a GRF register without paying a dependency penalty on each instruction. In this case, NoDDClr is used across an instruction sequence to allow the first instruction to set the destination dependency, interior instructions to write to the GRF register without dependency checks, and the last instruction to clear the dependency. (This sequence is referred to as a NoDDClr code block going forward). By only allowing the last instruction to clear the dependency, program execution is prevented from going beyond a certain point until all writes of that sequence are known to retire.

The problem arises if an exception is raised within a NoDDClr code block. In this case, there exists the potential for the System Routine to hang while attempting to save/restore a register used as a destination register by the NoDDCLr code block, as the outstanding dependency on that register will not clear until the final instruction of the NoDDCLr block is executed, sometime after the System Routine returns. Should the System Routine attempt to use that register, it hangs waiting on a dependency to be cleared by an instruction not yet issued.

Note: This is a known condition and will in some cases not allow the full GRF contents to be externally visible in System Routine scratch space during a break or halt exception.

To avoid this condition, guidelines are provided below for consideration. (Note that these are general guidelines, some of which can be alleviated through careful coding and register usage conventions and restrictions.)

Guideline
NoDDClr code blocks should only be used where absolutely necessary.
Instructions that may generate exceptions should not be placed within NoDDClr blocks. This includes most conditional branch instructions (if, do, while, ...).
If possible, use NoDDClr on registers high in the thread's register allocation (e.g. r120), thus even if a System Routine hang occurs, as much of the GRF is visible as possible. (Note that this would also require the System Routine to update the progress of the GRF dump, perhaps with each GRF block written, or to initialize the System Routine's scratch space to a known value, to be able to distinguish valid/locations from unwritten locations).

Also a driver implementation may consider a disable-NoDDClr option in which jitted code does not use the NoDDClr capability. In this case, there is no change to the code that is jitted other than removal of the NoDDClr instruction option. The code executes as normal, but with a higher number of thread switches in what would have been a NoDDClr code block.

Exception Descriptions

This section describes conditions that can cause exceptions and transfer control to the System Routine.

Illegal Opcode

The GEN ISA defines a single *illegal* opcode. The byte value of the *illegal* opcode is 0x00 due to it being a likely byte value encountered by a wayward instruction pointer value. The *illegal* instruction signals an exception if exception handling is enabled and invokes the system interrupt routine. If exception handling is NOT enabled, the illegal opcode is executed resulting in undetermined behavior including a system hang. Hardware decodes all legal opcodes supported. Any byte value that is not in the legal opcode list is decoded as an illegal opcode to trigger exception.

CALLA is not recognized as a legal opcode. When calla is used, illegal opcode exception must be disabled.

Undefined Opcodes

All undefined opcodes in the 8-bit opcode space (which includes instruction bit 7, reserved for future opcode expansion) are detected by hardware. If an undefined opcode is detected, the opcode is overridden by hardware, forcing the opcode value within the pipeline to the defined *illegal* opcode. The offending instruction, should it eventually be issued down the execution unit's pipeline, generates an Illegal Opcode exception as described in the section . The memory location of the offending opcode keeps its original value. That location can be queried to determine the opcode value.

Software Exception

A mechanism is provided to allow an application thread to invoke an exception and is triggered using the Software Exception Set and Clear bit of cr0.1. Sub-function determination and parameter passing into and out of the exception handler is left to convention between the system-thread and application-thread. The thread's IP is incremented before saving AIP and entering the System Routine, causing execution to resume at the next application-thread instruction after returning from the System Routine.

Context Save and Restore

The System Routine is also used to save and restore the context of the Execution Unit. This feature is enabled in GPGPU workloads *only*.

When the execution engine receives a preemption or an interrupt, the application thread invokes the System Routine. The System Routine is invoked only when all in-flight registers have retired. The system routine is used to save all the state of the EU to memory. When the sequence is complete, the master exception control bit is cleared. This action stops all execution for the given thread and invalidates the

thread. This means a new thread from a different context may be loaded. When the master exception control bit is cleared, software must ensure that all outstanding messages from the EU are dispatched out of the execution message pipeline. This is achieved by creating a dependency on the last send that is saving EU state. A dummy instruction before clearing the master exception control bit ensures that this is achieved.

The System Routine is also invoked on a context restore request. In this case a dummy thread is loaded into the EU which starts with the System Routine. This routine now restores the state of the EU. The restore sequence used in such a case should be consistent with the save sequence to ensure that state is restored correctly. After completing the restore sequence, the System Routine must clear the master exception control bit in the Control Register. This enables hardware to switch to the application thread which continues execution.

Programming Note	
Context:	Context Save and Restore
When context save and restore is required to be supported for GPGPU work loads, Stack Overflow exception handling will not be supported. Software will either need to ensure stack is either completely disabled OR used in such a way, an exception will not trigger.	

Events That Do Not Generate Exceptions

The conditions described in this section are either not recognized or do not generate an exception.

Illegal Instruction Format

This condition includes malformed instructions in which the opcode is legal, but the source or destination operands or other instruction attributes do not comply with the instruction specification. There is no direct hardware support to detect these cases and the outcome of issuing a malformed instruction is undefined.

Note that GEN does not support self-modifying code, therefore the driver has an opportunity to detect such cases before the thread is placed in service.

Malformed Message

A message's contents, destination registers, lengths, and descriptors are not interpreted in any way by the execution unit. Errors in specifying message fields do not raise exceptions in the EU but may be detected and reported by the shared functions.

GRF Register Out of Bounds

Unique GRF storage is allocated to each thread which, at a minimum, satisfies the register requirements specified in the thread's declaration. References to GRF register numbers beyond that called for in the thread's declaration do not generate exceptions. Depending on the implementation, out-of-bounds register numbers may be remapped to r0..r15, although this functionality should not be relied upon by the thread. The hardware guarantees the isolation of each thread's register space, thus there is no possibility of direct register manipulation via an out-of-bounds register access.

Hung Thread

There is no hardware mechanism in the EU to detect a hung thread and such a thread may remain hung indefinitely. It is expected that one or more hung threads will eventually cause the driver to recognize a context timeout and take appropriate recovery action.

Instruction Fetch Out of Bounds

The EU implements a full 32-bit instruction address range (with the 4 LSBs don’t care), making it possible for a thread to attempt to jump to any 16-byte aligned offset in the 32-bit instruction address range. (Instruction addresses are offsets from the General State Base Address.) The EU does not provide any type of address checking on instruction fetch requests sent to the memory/cache hierarchy, although error conditions for memory addresses are reported via the Page Table Error Register and other memory interface registers.

FPU Math Errors

The EU's floating point units (FPUs) have defined behaviors for traditional floating point errors and do not generate exceptions. There is no support for signaling FPU math errors as exceptions.

Adds the IEEE Exception Trap Enable bit, which enables trapping IEEE exception flags. If enabled, IEEE floating-point exceptions set sticky bits in the IEEE Exceptions field of sr0.1. Note that IEEE floating-point exceptions still do *not* transfer control to any handler.

Computational Overflow

Depending on source operand types and values, destination type, and the operation being performed, overflows may occur in the execution pipelines. Many instructions support the overflow (**.o**) conditional modifier that assigns flag bits based on whether or not an overflow occurs.

The EU never signals exceptions for overflows. Software must provide any overflow handling.

System Routine Example

The following code sequence illustrates some concepts of the System Routine. It is intended to be just a shell, without getting into the specifics of each exception handler.

The example frees enough MRF and GRF space to get the routine started, then jumps to the handler for the specific exception. Many other implementations are also valid, including single exception servicing (as opposed to looping) per invocation, and saving only the GRF or MRF space required by the exception being serviced.

```
#define ACC_DISABLE_MASK 0xFFFFFFFD
#define MASTER_EXCP_MASK 0x7FFFFFFF
#define SYSROUTINE_SCRATCH_BLKSIZE 16384 // for example

// Shared function IDs:
#define DPR 0x04000000
#define DPW 0x05000000

// Message lengths:
#define ML5 0x00500000
#define ML9 0x00900000

// Response lengths:
#define RL0 0x00000000
#define RL4 0x00040000
#define RL8 0x00080000

// Data port block sizes:
#define BS1_LOW  0x0000
#define BS1_HIGH 0x0100
#define BS2      0x0200
#define BS4      0x0300

// Scratch Layout:
 #define SCR_OFFSET_GRF 512          // + 16 MRF registers
#define SCR_OFFSET_ARF 512 + 4096  // + 16 MRF + 128 GRF registers

// Write data port constants:
// target=dcache, type= oword_block_wr, binding_tbl_offset=0
#define DPW 0x000

// Read data port constants:
// target=dcache, type= oword_block_rd, binding_tbl_offset=0
#define DPR 0x000

Sys_Entry:  // Entry point to the System Routine.

// Disable accumulator for system routine:
and (1) cr0.0 cr0.0 ACC_DISABLE_MASK {NoMask}

// Calc scratch offset for this thread into r0.4:
shr (1) r0.4 sr0.0:uw 6 {NoMask}
add (1) r0.4 r0.4 sr0.0:ub {NoMask}
mul (1) r0.4 r0.4 SYSROUTINE_SCRATCH_BLKSIZE {NoMask}

// Setup m0 with block offset:
mov (8) m0 r0{NoMask}
```

```
// Save r0..r1 to system scratch:
// Note: done as a single register to guarantee external visibility
// See Use of NoDDClr mov (16) m1 r0 {NoMask}
send (8) m0 null null DPW|ML2|RL0 {NoMask}

// Save r2..r3 to free some room:
mov (16) m3 r2 {NoMask}
add (1) m0.2 r0.4 SCR_OFFSET_GRF + 64 {NoMask}
send (8) m0 null null DPW|ML4|RL0 {NoMask}

// Save r4..r7 to free some room (optional, depending on needs):
mov (16) m8 r4 {NoMask}
mov (16) m10 r6 {NoMask}
add (1) m7.2 r0.4 (SCR_OFFSET_GRF + 128) {NoMask}
send (8) m7 null null DPW|ML5|RL0 {NoMask}

// Save r8..r11 to free some room (optional, depending on needs):
mov (16) m1 r8 {NoMask}
mov (16) m3 r10 {NoMask}
add (1) m0.2 r0.4 (SCR_OFFSET_GRF + 256) {NoMask}
send (8) m0 null null DPW|ML5|RL0 {NoMask}

// Save r12..r15 to free some room (optional, depending on needs):
mov (16) m8 r12 {NoMask}
mov (16) m10 r14 {NoMask}
add (1) m7.2 r0.4 (SCR_OFFSET_GRF + 384) {NoMask}
send (8) m7 null null DPW|ML5|RL0 {NoMask}

// Save selected ARF registers (optional, depending on use):
// flags, others ...
// Save f0.0:
mov (1) r1.0:uw f0.0 {NoMask}

Next:  // Exceptions pending? If not, exit.

cmp.e (1) f0.0 cr0.4:uw 0:uw {NoMask}
(f0.0) mov (1) IP EXIT {NoMask}

// Find highest priority exception:
lzd (1) r1.1:uw cr0.4:uw {NoMask}

// Jump table to service routine:
jmpi (1) r1.1:uw{NoMask}
mov (1) IP CRService_0 {NoMask}
mov (1) IP CRService_1 {NoMask}
mov (1) IP CRService_2 {NoMask}
...
mov (1) IP CRService_15{NoMask}
mov (1) IP Next

Service_0:
// Clear exception from cr0.1.
// Perform service routine.
// Jump to exit (or if looping on exceptions, go to next loop).
...
Service_15:
// Clear exception from cr0.1.
// Perform service routine.
// Jump to exit (or if looping on exceptions, go to next loop).

Exit:
// Restore f0.0.
```

```
// Restore other ARF registers (as required).
// Restore r12..r15.
// Restore r8..r11.
// Restore r4..r7.
// Restore r0..r3.
// Restore m8..m15.
// Restore m0..m7.
// Clear Master Exception State bit in cr0.0:
and (1) cr0.0 cr0.0 MASTER_EXCP_MASK
nop (16)
```

Below is a code sequence to programmatically clear the GRF scoreboard in case of a timeout waiting on a register that may never return.

At this point, all we know is we have a hung thread. We'd like to copy the GRF to scratch memory to make it visible, but there may be a register that is hung with an outstanding dependency. To get around any hung dependency, walk the GRF using NoDDChk, using an execution mask of f0 == 0 so we don't touch the register contents.

```
Clear_Dep:
mov f0 0x00
(f0) mov r0 0x00 {NoDDChk}
(f0) mov r1 0x00 {NoDDChk}
(f0) mov r2 0x00 {NoDDChk}
...
(f0) mov r127 0x00 {NoDDChk}
// GRF scoreboard now cleared. Do I delete the entire highlighted text?
```

Instruction Set Summary

This topic is currently under development.

Instruction Set Characteristics

This topic is currently under development.

SIMD Instructions and SIMD Width

GEN instructions are SIMD (single instruction multiple data) instructions. The number of data elements per instruction, or the execution size, depends on the data type. For example, the execution size for GEN instructions operating on 256-bit wide vectors can be up to 8 for 32-bit data types, and be up to 16 for 16-bit data. The maximum execution size for GEN instructions for 8-bit data types is also limited to 16.

An instruction compression mode is supported for 32-bit instructions (including mixed 32-bit and 16-bit data computation). A compressed GEN instruction works on twice as much SIMD data as that for a non-compressed GEN instruction. A compressed instruction is converted into two native instructions by the instruction dispatcher in the EU.

GEN instructions are executed on a narrower SIMD execution pipeline. Therefore, GEN native instructions take multiple execution cycles to complete.

Instruction Operands and Register Regions

Most GEN instructions may have up to three operands, two sources and one destination. Each operand is able to address a register region. Source operands support negate and absolute modifier and channel swizzle, and the destination operand supports channel mask.

Dual destination instructions are also supported (four-operand instructions in a general sense): One case is for the implied destination – flag register, where the conditional modifiers and the predicate modifiers may apply. Another case is the message header creation (implied move or implied assembling of the header) in the *send* instruction.

Each execution channel contains an accumulator that is wider than the input data to support back-to-back accumulation operations with increased precision. The added precision (see accumulator register description in Execution Environment chapter) determines the maximum number of accumulations before possible overflow. The accumulator can be pre-loaded through the use of *mov*. It can also be pre-loaded by arithmetic instructions such as *add* or *mul*, since the result of these instructions can go to the accumulator. The accumulator registers are per thread and therefore safe for thread switching.

Register access can be direct or register-indirect. Register-indirect register access uses address registers plus an immediate offset term to compute the register addresses, and only applies to the first source operand (src0) and/or the destination operand.

There is one address register.

Description
There are 16 address sub-registers.

Each sub-register contains a 16-bit unsigned value. The leading two sub-registers form a special doubleword that can be used as the descriptor for the send instruction.

Source operand can also be immediate value (also referred to as inline constants). For instructions with two source operands, only the second operand src1 is allowed to be immediate. For instructions with only one source operand, the source operand src0 is used and it can be an immediate.

An immediate source operand can be a scalar value of specified type up to 32-bit wide, which is replicated to create a vector with length of Execution Size. An immediate operand can also be a special 32-bit vector with 8 elements each of 4-bit signed integer value, or a 32-bit vector with 4 elements each of 8-bit restricted float value.

Instruction Execution

It is implied that all instructions operate across all channels of data unless otherwise specified either via destination mask, predication, execution mask (caused by SIMD branch and loop instructions), or execution size.

Instruction execution size can be specified per instruction, from scalar (*ExecSize* = 1) up to the maximal execution size supported for the data type, with the restriction that execution size can only be in power of 2.

Instruction Formats

This section shows the machine formats of the GEN instruction set. The instructions in the GEN architecture have a fixed length of 128 bits in the native format. A compact format, discussed separately in this volume, can represent some instructions using 64 bits. Out of the 128 bits in the native format, there are 120 bits in use, and the remaining bits are reserved for future extensions. One instruction consists of instruction fields that control various stages of execution. These fields are roughly grouped into the 4 DWords as follows:

- Instruction Operation Doubleword (DW0) contains the Opcode and other general instruction control fields.
- Instruction Destination Doubleword (DW1) specifies the destination operand (dst) and the register file and type of source operands.
- Instruction Source 0 Doubleword (DW2) contains the first source operand (src0).
- Instruction Source 1 Doubleword (DW3) contains the second source operand (src1) and is used to hold any 32-bit immediate source (Imm32 as src0 or src1).

Instructions with one source operand of type DF, Q, or UQ can use an Imm64 64-bit immediate source operand in DWord 2 and DWord3.

Most instructions have 1 or 2 source operands and use a common instruction format. Within that format, there are variations based on AddrMode and AccessMode. There is a separate instruction

format for a small number of instructions with 3 source operands. Send, math, and branching instructions have format variations described separately.

The 3-source instructions have the following restrictions:

- Only GRF registers can be sources and only GRF registers can be the destination.
- Subregister numbers have DWord granularity if AccessMode is Align16.
- AccessMode is Align16, uses Align16-style swizzling, with extra replication control. There is no other regioning support.

The next two subsections describe the instruction formats for various processor generations using tables. The following diagrams provide another view of the same information. The first diagram is for native instructions with one or two source operands.

GEN Instruction Format – 1-src and 2-src [CHV, BSW]

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	AddrMode = Direct		AddrMode = Indirect		SEND		MATH	Branch (2offsets)	Branch (1offset)	Imm Src[31:0]	Imm Src[63:0]
					AccessMode = Align16	AccessMode = Align1	AccessMode = Align16	AccessMode = Align1	MsgDesc Imm	MsgDesc Reg					
	1	127	127	1					EOT						
	6	126	121	6											
	4	120	117	4	Src1.VertStride										
	1	116	116	1											
	2	115	114	2		Src1.Width		Src1.Width							
	2	113	112	2	Src1.ChanSel[7:4]	Src1.HorzStride	Src1.ChanSel[7:4]	Src1.HorzStride							
	1	111	111	1	Src1.AddrMode										
	2	110	109	2	Src1.SrcMod										
	4	108	105	4			Src1.AddrSubRegNum								
	4	104	101	4	Src1.RegNum [7:0]										
	1	100	100	1	Src1.SubRegNum [4]		Src1.AddrImm[8:4]								
3	4	99	96	4	Src1.ChanSel[3:0]	Src1.SubRegNum [4:0]	Src1.ChanSel[3:0]	Src1.AddrImm[8:0]			Same	JIP[31:0]	JIP[31:0]	Imm[31:0]	
	1	95	95	1											
	4	94	91	4	Src1.SrcType										
	2	90	89	2	Src1.RegFile										
	4	88	85	4	Src0.VertStride										
	1	84	84	1											
	2	83	82	2		Src0.Width		Src0.Width							
	2	81	80	2	Src0.ChanSel[7:4]	Src0.HorzStride	Src0.ChanSel[7:4]	Src0.HorzStride							
	1	79	79	1	Src0.AddrMode										
	2	78	77	2	Src0.SrcMod										
	4	76	73	4			Src0.AddrSubRegNum								
	4	72	69	4	Src0.RegNum [7:0]		Src0.AddrImm [8:4]								
	1	68	68	1	Src0.SubRegNum [4]										
2	4	67	64	4	Src0.ChanSel[3:0]	Src0.SubRegNum [4:0]	Src0.ChanSel[3:0]	Src0.AddrImm [8:0]				UIP[31:0]			Imm[63:0]
	1	63	63	1	Dst.AddrMode										
	2	62	61	2		Dst.HorzStride		Dst.HorzStride							
	4	60	57	4			Dst.AddrSubRegNum								
	4	56	53	4	Dst.RegNum [7:0]										
	1	52	52	1	Dst.SubRegNum [4]		Dst.AddrImm [8:4]								
	4	51	48	4	Dst.ChanEn[3:0]	Dst.SubRegNum [4:0]	Dst.ChanEn[3:0]	Dst.AddrImm [8:0]							
	1	47	47	1	Reserved										
	4	46	43	4	Src0.SrcType										
	2	42	41	2	Src0.RegFile										
	4	40	37	4	Dst.Type										
	2	36	35	2	Dst.RegFile										
	1	34	34	1	WECtrl										
	2	33	32	2	FlagRegNum/FlagSubRegNum				Same		Same	Same	Same	Same	Same
	1	31	31	1	Saturate										
	1	30	30	1	DebugCtrl										
	1	29	29	1	CmptCtrl										
	1	28	28	1	AccWrCtrl / BranchCtrl				Same		Same	Same	Same		
	4	27	24	4	CondModifier				SFID[3:0]		FC[3:0]	MBZ	MBZ		
	3	23	21	3	ExecSize										
	1	20	20	1	PredInv										
	4	19	16	4	PredCtrl										
	2	15	14	2	ThreadCtrl										
	2	13	12	2	QtrCtrl										
	1	11	11	1	NibCtrl										
	2	10	9	2	DepCtrl										
	1	8	8	1	AccessMode				Same		Same	Same	Same	Same	Same
	1	7	7	1	(reserved for Opcode)										
0	7	6	0	7	Opcode				Same		Same	Same	Same	Same	Same

The next diagram is for native instructions with three source operands.

GEN Instruction Format – 3-src [CHV, BSW]

DW#	Instr Bits	High Bit	Low Bit	Instr Bits	Description
	1	127	127	1	*reserved*
	1	126	126	1	*reserved*
	8	125	118	8	Src2 Regnum
	3	117	115	3	Src2 Subregnum
	8	114	107	8	Src2 Swizzle
	1	106	106	1	Src2 RepCtrl
	1	105	105	0	*reserved*
	8	104	97	8	Src1 Regnum
	3	96	94	3	Src1 Subregnum
	8	93	86	8	Src1 Swizzle
	1	85	85	1	Src1 RepCtrl
	1	84	84	0	*reserved*
	8	83	76	8	Src0 Regnum
	3	75	73	3	Src0 Subregnum
	8	72	65	8	Src0 Swizzle
2,3	1	64	64	1	Src0 RepCtrl
	8	63	56	8	Dst Regnum
	3	55	53	3	Dst Subregnum
	4	52	49	4	Dst chan enable
	3	48	46	3	Dst Type
	3	45	43	3	Src Type
	2	42	41	2	Src2 Modifier
	2	40	39	2	Src1 Modifier
	2	38	37	2	Src0 Modifier
	1	36	36	1	*reserved*
	1	35	35	1	*reserved*
	1	34	34	1	*WECtrl*
1	2	33	32	2	*FlagRegNum/FlagSubRegNum*
	1	31	31	1	*Saturate*
	1	30	30	1	*DebugCtrl*
	1	29	29	1	*CmptCtrl*
	1	28	28	1	*AccWrCtrl*
	4	27	24	4	*CondModifier*
	3	23	21	3	*ExecSize*
	1	20	20	1	*PredInv*
	4	19	16	4	*PredCtrl*
	2	15	14	2	*ThreadCtrl*
	2	13	12	2	*QtrCtrl*
	1	11	11	1	*NibCtrl*
	2	10	9	2	*DepCtrl*
	1	8	8	1	*AccessMode*
	1	7	7	0	*(reserved for Opcode)*
0	7	6	0	7	*Opcode*

GEN Instruction Format – 3-src [CHV, BSW]

DW#	Instr Bits	High Bit	Low Bit	Instr Bits	Description
	1	127	127	1	*reserved*
	1	126	126	1	Src2 Subregnum
	8	125	118	8	Src2 Regnum
	3	117	115	3	Src2 Subregnum
	8	114	107	8	Src2 Swizzle
	1	106	106	1	Src2 RepCtrl
	1	105	105	1	Src1 Subregnum
	8	104	97	8	Src1 Regnum
	3	96	94	3	Src1 Subregnum
	8	93	86	8	Src1 Swizzle
	1	85	85	1	Src1 RepCtrl
	1	84	84	1	Src0 Subregnum
	8	83	76	8	Src0 Regnum
	3	75	73	3	Src0 Subregnum
	8	72	65	8	Src0 Swizzle
2,3	1	64	64	1	Src0 RepCtrl
	8	63	56	8	Dst Regnum
	3	55	53	3	Dst Subregnum
	4	52	49	4	Dst chan enable
	3	48	46	3	Dst Type
	3	45	43	3	Src Type
	2	42	41	2	Src2 Modifier
	2	40	39	2	Src1 Modifier
	2	38	37	2	Src0 Modifier
	1	36	36	1	Src1 Type
	1	35	35	1	Src2 Type
	1	34	34	1	*WECtrl*
1	2	33	32	2	*FlagRegNum/FlagSubRegNum*
	1	31	31	1	*Saturate*
	1	30	30	1	*DebugCtrl*
	1	29	29	1	*CmptCtrl*
	1	28	28	1	*AccWrCtrl*
	4	27	24	4	*CondModifier*
	3	23	21	3	*ExecSize*
	1	20	20	1	*PredInv*
	4	19	16	4	*PredCtrl*
	2	15	14	2	*ThreadCtrl*
	2	13	12	2	*QtrCtrl*
	1	11	11	1	*NibCtrl*
	2	10	9	2	*DepCtrl*
	1	8	8	1	*AccessMode*
	1	7	7	0	*(reserved for Opcode)*
0	7	6	0	7	*Opcode*

Native Instruction Layouts

This section describes the Execution Unit instruction formats.

DWord 0, bits 31:0 of the 128-bit instruction, has the same format regardless of the number of source operands.

The following three tables cover the most common instruction format, for instructions with 1 or 2 source operands; then the format for the few instructions with 3 source operands; and finally format variations used by a few specific instructions, including branching instructions.

Execution Unit Instruction Format for 1 or 2 Source Operands CHV, BSW

Bits	Description	AddrMode and AccessMode Variations			
		AddrMode = Direct		AddrMode = Indirect	
		Align16	Align1	Align16	Align1
Any Imm32 32-bit or Imm64 64-bit immediate operand uses bits 127:96, replacing the following fields.					
127:122	Reserved				
121	Varies based on AddrMode	Reserved		Src1.AddrImm[9]	
120:117	Src1.VertStride				
116	Varies based on AccessMode	Reserved	Src1.Width	Reserved	Src1.Width
115:114		Src1.ChanSel[7:4]		Src1.ChanSel[7:4]	
113:112				Src1.HorzStride	
111	Src1.AddrMode				
110:109	Src1.SrcMod				
108:105	Src1.SrcMod	Src1.RegNum		Src1.AddrSubRegNum	
104:101				Src1.AddrImm[8:4]	Src1.AddrImm[8:0]
100					
99:96		Src1.ChanSel[3:0]	Src1.ChanSel[3:0]		
Any Imm64 64-bit immediate operand uses bits 127:64, replacing the following fields.					
95	Varies based on AddrMode	Reserved		Src0.AddrImm[9]	
94:91	Src1.SrcType				
90:89	Src1.RegFile				
88:85	Src0.VertStride				
84	Varies based on AccessMode	Reserved	Src0.Width	Reserved	Src0.Width
83:82		Src0.ChanSel[7:4]		Src0.ChanSel[7:4]	

Bits	Description	AddrMode and AccessMode Variations			
		AddrMode = Direct		AddrMode = Indirect	
		Align16	Align1	Align16	Align1
81:80				Src0.HorzStride	
79	Src0.AddrMode				
78:77	Src0.SrcMod				
76:73	Varies based on AddrMode and AccessMode	Src0.RegNum		Src0.AddrSubRegNum	
72:69				Src0.AddrImm[8:4]	Src0.AddrImm[8:0]
68					
67:64		Src0.ChanSel[3:0]	Src0.ChanSel[3:0]		
63	Dst.AddrMode				
62:61	Varies based on AccessMode	Reserved	Dst.HorzStride	Reserved	Dst.HorzStride
60:57	Varies based on AddrMode and AccessMode	Dst.RegNum		Dst.AddrSubRegNum	
56:53				Dst.AddrImm[8:4]	Dst.AddrImm[8:0]
52					
51:48		Dst.ChanEn[3:0]	Dst.ChanEn[3:0]		
47	Varies based on AddrMode	Reserved : MBZ		Dst.AddrImm[9]	
46:43	Src0.SrcType				
42:41	Src0.RegFile				
40:37	Dst.DstType				
36:35	Dst.RegFile				
34	MaskCtrl				
33:32	FlagRegNum / FlagSubRegNum				
31	Saturate				
29	CmptCtrl				
28	AccWrCtrl				
27:24	CondModifier				
23:21	ExecSize				
20	PredInv				
19:16	PredCtrl				
15:14	ThreadCtrl				
13:12	QtrCtrl				

Bits	Description	AddrMode and AccessMode Variations			
		AddrMode = Direct		AddrMode = Indirect	
		Align16	Align1	Align16	Align1
11	NibCtrl				
10:9	DepCtrl				
8	AccessMode				
7	Reserved (for future Opcode expansion)				
6:0	Opcode				

The 3-source operand instructions are:

1. *bfe* - Bit Field Extract
2. *bfi2* - Bit Field Insert 2
3. *lrp* - Linear Interpolation
4. *mad* - Multiply Add
5. *madm* - Multiply Add for Macro

In the 3-source instruction format, the upper QWord contains three groups of 21 bits for the three source operands, where each group contains four fields in 20 bits and otherwise adjacent groups are separated by single reserved bits.

Specific instructions have different instruction formats as described below. These instructions include send / sendc, math, and branch instructions.

Execution Unit Instruction Format for Specific Instructions [CHV, BSW]

Bits	Regular 1 or 2 Source Operands Description	Empty white areas mean Same, use the regular description		
		send / sendc	math	Branch Instructions
127	Reserved	EOT		JIP[31:0]
126:125	Reserved	Imm[28:0] / Reg32		
124:121	Reserved			
120:117	Src1.VertStride			
116:112	Varies based on AccessMode			
111	Src1.AddrMode			
110:109	Src1.SrcMod			
108:96	Varies based on AddrMode and AccessMode			
95	Reserved			UIP[31:0] (2-offset branches)
94:91	Src1.SrcType			
90:89	Src1.RegFile			
88:85	Src0.VertStride			
84:80	Varies based on AccessMode			
79	Src0.AddrMode			
78:77	Src0.SrcMod			
76:64	Varies based on AddrMode and AccessMode			
63	Dst.AddrMode			
62:61	Varies based on AccessMode			
60:48	Varies based on AddrMode and AccessMode			
47	Reserved			
46:43	Src0.SrcType			
42:41	Src0.RegFile			
40:37	Dst.DstType			
36:35	Dst.RegFile			
34	MaskCtrl			

33:32	FlagRegNum / FlagSubRegNum			
31	Saturate			
30				
29	CmptCtrl			
28	AccWrCtrl			
27:24	CondModifier	SFID[3:0]	FC[3:0]	MBZ
23:21	ExecSize			
20	PredInv			
19:16	PredCtrl			
15:14	ThreadCtrl			
13:12	QtrCtrl			
11	NibCtrl			
10:9	DepCtrl			
8	AccessMode			
7	Reserved (for future Opcode expansion)			
6:0	Opcode			

Instruction Fields

This section contains two large tables that together describe all instruction fields. The first table describes common instruction fields that are not specific to a source or destination operand. The second table describes fields used to describe source or destination operands, including fields that describe register regioning and immediate operand fields.

Notable changes for CHV, BSW instructions include adding three new data types (Q, UQ, HF), widening all type fields to four bits each to accomodate the new type encodings, providing a new source modifier interpretation for logical operations, widening the AddrSubRegNum fields to four bits to accomodate 16 (rather than 8) address subregisters, and supporting 64-bit immediate (Imm64) source values for instructions with single source operands.

In the assembler syntax, some fields appear in the positions used for destination or source operands but are not normal operands. Such fields appear in the Common Instruction Fields table, notably the JIP and UIP instruction offsets used in some flow control instructions.

Common Instruction Fields (Alphabetically by Short Name) [CHV, BSW]

Field	Description
AccessMode	**Access Mode.** This field controls operand access for the instruction. It applies to all source and destination operands. When it is clear (**Align1**), the instruction uses byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are not supported. Elements must still be aligned on element-size boundaries. For example, a DWord operand must be aligned on a 4-byte boundary. When it is set (**Align16**), the instruction uses 16-byte-aligned addressing for source and destination operands. Source swizzle control and destination mask control are supported in this mode. A register region must start on 16-byte aligned address, but individual elements do not. For example, a packed array of 16 bytes can be an operand, where only the first byte, at offset 0, is at a 16-byte aligned address. 0 = **Align1** 1 = **Align16**
AccWrCtrl/ BranchCtrl	**Accumulator Write Control.** Enable or disable implicitly writing results to the accumulator as well as to the destination. 0 = Do not write results to the accumulator; write results only to the destination. 1 = **AccWrEn.** Write results to the accumulator as well as to the destination. This bit should not be set if the accumulator is the explicit destination operand. Some instructions do not allow this option, including *mov* and *send*. **Branch Control.** Used by the *goto* instruction to control branching. See the **goto** instruction description for more information about BranchCtrl.
CmptCtrl	**Compaction Control.** Indicates whether the instruction is compacted to the 64-bit compact instruction format. When this bit is set, the 64-bit compact instruction format is used. The EU decodes the compact format using lookup tables internal to the hardware, but documented for use by software tools. Only some instruction variations can be compacted, the variations supported by those lookup tables and the compact format. See EU Compact Instruction Format [CHV, BSW] for more information. 0 = No Compaction. No compaction. 128-bit native instruction supporting all instruction options. 1 = **CmptCtrl**. Compaction is enabled. 64-bit compact instruction supporting only some instruction variations.
CondModifier	**Condition Modifier.** This field sets the flag register based on internal

Field	Description
	conditional signals output from the execution pipe such as sign, zero, overflow, NaNs, etc. If this field is 0000b, no flag registers are updated. Flag registers are not updated for instructions with embedded compares. This field applies to all instructions except send, sendc, and math. 0000b = Do not modify the flag register (normal) 0001b = Zero or Equal (**.z** or **.e**) 0010b = Not Zero or Not Equal (**.nz** or **.ne**) 0011b = Greater than (**.g**) 0100b = Greater than or equal (**.ge**) 0101b = Less than (**.l**) 0110b = Less than or equal (**.le**) 0111b = Reserved 1000b = Overflow (**.o**) 1001b = Unordered with Computed NaN (**.u**) 1010b-1111b = Reserved
DepCtrl	**Destination Dependency Control.** This field selectively disables destination dependency check and clear for this instruction. When it is set to 00b, normal destination dependency control is performed for the instruction; hardware checks for destination hazards to ensure data integrity. Specifically, a destination register dependency check is conducted before the instruction is made ready for execution. After the instruction is executed, the destination register scoreboard is cleared when the destination operands retire. When **NoDDClr** is set, the destination register scoreboard is *not* cleared when destination operands retire. When this field is not 00b, hardware does not protect against destination hazards for the instruction. These settings are typically used to assemble data in a fine grained fashion (for example, a matrix-vector compute with dot-product instructions), where data integrity is guaranteed by software based on the intended usage of instruction sequences. 00b = Destination dependency checked and cleared (normal). 01b = **NoDDClr**. Destination dependency checked but not cleared. 10b = **Reserved** 11b = **NoDDClr**. Destination dependency not cleared. See the Destination Hazard and the Use of NoDDClr sections for more information.
EOT	**End of Thread.** For a *send* or *sendc* instruction, this bit controls thread

Field	Description
	termination. It is not used for other instructions. For a *send* or *sendc*instruction, if this field is set, the EU terminates the thread and also sets the EOT bit in the message sideband. 0 = The thread is not terminated. 1 = End of thread (EOT).
ExDesc	**Extended message descriptor.** A 32 bit immediate extended message descriptor for *send* and *sendc* instructions. This field is not used for other instructions.
ExecSize	**Execution Size.** Specifies the number of parallel execution channels and data elements for the instruction, a power of 2 from 1 to 32. The size cannot exceed the maximum number of channels allowed for the largest data type specified for a source or destination operand. *ExecSize* x largest data type size in bytes <= 64. 000b = 1 Channels (scalar operation). 001b = 2 Channels. Any data type. 010b = 4 Channels. Any data type. 011b = 8 Channels. Any data type. 100b = 16 Channels. 4-byte or smaller data types. Excludes DF, Q, and UQ types. 101b = 32 Channels. 2-byte or 1-byte data types. Excludes D, DF, F, Q, UD, and UQ types. 110b-111b = Reserved. An operand's *Width* must be <= *ExecSize*.
FC	**Function Control.** Specifies the extended math function carried out by the *math*instruction. Not used for any other instruction. This field is in the same position as the CondModifier instruction field, so the *math* instruction does not support conditional modifiers. See the *math* Extended Math Function [CHV, BSW] instruction description for more information about this field.
FlagRegNum/ **FlagSubRegNum**	**Flag Register Number/Flag Sub-Register Number.** Selects the 32-bit flag register (f0 or f1) and the 16-bit flag subregister (.0 or .1). The specified flag subregister is the source for any predication and the destination for new flag values produced by any enabled conditional modifier. A flag subregister can be both a predication source and a conditional modifier destination in the same instruction. The number of flag bits used or updated depends on the execution size.

Field	Description
	00b f0.0 01b f0.1 10b f1.0 11b f1.1
JIP	**Jump Instruction Pointer.** A Doubleword Signed Integer offset relative to the current IP (which references the current instruction) in units of bytes. Typically an immediate value in the instruction. For the *brc* (Branch Converging) instruction, both JIP and UIP can be contained in a register.
MaskCtrl	**Mask Control** (formerly WECtrl/Write Enable Control). This flag disables the normal write enables; it should normally be 0. 0 = Use the normal write enables in Dst.ChanEn (normal setting). 1 = NoMask. Write all channels except those disabled by predication or by other masks besides the write enables. MaskCtrl = NoMask also skips the check for PcIP[n] == ExIP before enabling a channel, as described in the Evaluate Write Enable section.
NibCtrl	**Nibble Control.** This field is used in some instructions along with QtrCtrl. See the description of QtrCtrl below. NibCtrl is only used for SIMD4 instructions with a 64bit datatype as source or destination. 0 = Use an odd 1/8th for DMask/VMask and ARF (first, third, fifth, or seventh depending on QtrCtrl). 1 = Use an even 1/8th for DMask/VMask and ARF (second, fourth, sixth, or eighth depending on QtrCtrl). Note that if eighths are given zero-based indices from 0 to 7, then NibCtrl = 0 indicates even indices and NibCtrl = 1 indicates odd indices.
OpCode	**Operation Code.** Contains the instruction operation code. Each opcode used is given a unique mnemonic. For example, the opcode 0x01 has the mnemonic *mov* indicating a Move instruction. See the Opcode Encoding section for details of opcode encoding.
PredCtrl	**Predicate Control.** This field, together with PredInv, controls generating the predication mask (PMask) for the instruction. It allows per-channel conditional execution the instruction based on the content of the selected flag register. The encoding depends on the access mode. See the Predication section for more information about predication. In **Align16** access mode, there are eight encodings (including no predication). All encodings are based on group-of-4 predicate bits, including channel sequential, replication swizzles and horizontal any or all operations. The same

Field	Description
	configuration is repeated for each group-of-4 execution channels. In **Align1** access mode, there are twelve encodings (including no predication). The encodings apply to all execution channels with explicit channel grouping from a single channel up to a group of 16 channels. Predicate Control in **Align16** access mode: 0000b = No predication (normal). 0001b = Predication with sequential flag channel mapping. 0010b = Predication with replication swizzle **.x**. 0011b = Predication with replication swizzle **.y**. 0100b = Predication with replication swizzle **.z**. 0101b = Predication with replication swizzle **.w**. 0110b = Predication with **.any4h.** 0111b = Predication with **.all4h.** 1000b-1111b = Reserved. Predicate Control in **Align1** access mode: 0000b = No predication (normal). 0001b = Predication with sequential flag channel mapping. 0010b = Predication with **.anyv** (f0.0 OR f0.1 on the same channel). 0011b = Predication with **.allv** (f0.0 AND f0.1 on the same channel). 0100b = Predication with **.any2h** (any in group of 2 channels). 0101b = Predication with **.all2h** (all in group of 2 channels). 0110b = Predication with **.any4h** (any in group of 4 channels). 0111b = Predication with **.all4h** (all in group of 4 channels). 1000b = Predication with **.any8h** (any in group of 8 channels). 1001b = Predication with **.all8h** (all in group of 8 channels). 1010b = Predication with **.any16h** (any in group of 16 channels). 1011b = Predication with **.all16h** (all in group of 16 channels). 1100b = Predication with **.any32h** (any in group of 32 channels). 1101b = Predication with **.all32h** (all in group of 32 channels). 1110b-1111b = Reserved.
PredInv	**Predicate Inverse.** Together with *PredCtrl*, controls generation of the predication mask (PMask) for the instruction. When it is set and PredCtrl is not 0000b, predication uses the inverse of the predication bits produced by

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>PredCtrl. In other words, the effect of PredInv happens after the effects of PredCtrl.

This field is ignored if PredCtrl is 0000b; there is no predication.

0 = +. Positive polarity of predication. Use the predication mask produced by PredCtrl.

1 = –. Negative polarity of predication. If PredCtrl is nonzero, invert the predication mask.

PMask is the final predication mask produced by the effects of both fields.</td></tr>
<tr><td>QtrCtrl</td><td>Quarter Control. This field provides explicit control for ARF selection.

This field combined with ExecSize determines which channels are used for the ARF registers.

Along with NibCtrl, 1/8 DMask/VMask and ARF can be selected.
<table>
<tr><th>QtrCtrl</th><th>NibCtrl</th><th>ExecSize</th><th>Syntax</th><th>Description</th></tr>
<tr><td>00b</td><td>0</td><td>8</td><td>1Q</td><td>Use first quarter for DMask/VMask.
Use first half for everything else.</td></tr>
<tr><td>01b</td><td>0</td><td>8</td><td>2Q</td><td>Use second quarter for DMask/VMask.
Use second half for everything else.</td></tr>
<tr><td>10b</td><td>0</td><td>8</td><td>3Q</td><td>Use third quarter for DMask/VMask.
Use first half for everything else.</td></tr>
<tr><td>11b</td><td>0</td><td>8</td><td>4Q</td><td>Use fourth quarter for DMask/VMask.
Use second half for everything else.</td></tr>
<tr><td>00b</td><td>0</td><td>16</td><td>1H</td><td>Use first half for DMask/VMask.
Use all channels for everything else.</td></tr>
<tr><td>10b</td><td>0</td><td>16</td><td>2H</td><td>Use second half for DMask/VMask.
Use all channels for everything else.</td></tr>
<tr><td>00b</td><td>0</td><td>4</td><td>1N</td><td>Use first 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>00b</td><td>1</td><td>4</td><td>2N</td><td>Use second 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>01b</td><td>0</td><td>4</td><td>3N</td><td>Use third 1/8th for DMask/VMask and ARF.</td></tr>
</table></td></tr>
</table>

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td></td><td>
<table>
<tr><td>01b</td><td>1</td><td>4</td><td>4N</td><td>Use fourth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>10b</td><td>0</td><td>4</td><td>5N</td><td>Use fifth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>10b</td><td>1</td><td>4</td><td>6N</td><td>Use sixth 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>11b</td><td>0</td><td>4</td><td>7N</td><td>Use seventh 1/8th for DMask/VMask and ARF.</td></tr>
<tr><td>11b</td><td>1</td><td>4</td><td>8N</td><td>Use eighth 1/8th for DMask/VMask and ARF.</td></tr>
</table>
2H is only allowed for SIMD16 instruction in Single Program Flow mode (SPF = 1).

NibCtrl is only allowed for SIMD4 instructions with a 64bit datatype as source or destination type.
</td></tr>
<tr><td>Reg32</td><td>In a send or sendc instruction refers to the option of providing the message descriptor field DWord, of which bits 28:0 are used, in the first two words of the Address Register rather than as an immediate operand.</td></tr>
<tr><td>Saturate</td><td>Saturate. Enables or disables destination saturation.

When it is set, output values to the destination register are saturated. The saturation operation depends on the destination data type. Saturation is the operation that converts any value outside the saturation target range for the data type to the closest value in the target range.

For a floating-point destination type, the saturation target range is [0.0, 1.0]. For a floating-point NaN, there is no “closest value”; any NaN saturates to 0.0. Note that enabling Saturate overrides all of the NaN propagation behaviors described for various numeric instructions. Any floating-point number greater than 1.0, including +INF, saturates to 1.0. Any negative floating-point number, including -INF, saturates to 0.0. Any floating-point number in the range 0.0 to 1.0 is not changed by saturation.

For an integer destination type, the maximum range for that type is the saturation target range. For example, the saturation range for B (Signed Byte Integer) is [-128, 127].

When Saturate is clear, destination values are not saturated. For example, a wrapped result (modulo) is output to the destination for an overflowed integer value. See the Numeric Data Types section for information about data types and their ranges.

0 = No destination modification (normal).

1 = sat. Saturate the output.</td></tr>
</table>

Field	Description
SFID	**Shared Function ID.** Specifies a shared function that is the target of a *send*or *sendc* instruction. This field is not used for any other instructions. This field is in the same position as the CondModifier instruction field, so the *send* and *send*cinstructions do not support conditional modifiers.
ThreadCtrl	**Thread Control.** This field provides explicit control for thread switching. If this field is set to 00b, it is up to the Execution Unit to manage thread switching. This is the normal operating mode. In this mode, for example, if the current instruction cannot proceed due to operand dependencies, the EU switches to the next available thread to fill the compute pipe. For another example, if the current instruction is ready to run but another thread with higher priority also has an instruction ready, the EU switches to that thread. If this field is **Switch**, a forced thread switch occurs after the current instruction executes and before the next instruction. In addition, a long delay (longer than the execution pipe latency) for the current thread is introduced for the thread. Particularly, the instruction queue of the current thread is flushed after the current instruction is dispatched for execution. If this field is **Atomic**, the next instruction gets highest priority in the thread arbitration for the execution pipelines. **Switch** is designed primarily as a safety feature in case there are race conditions for certain instructions. 00b = Normal Thread Control. Execution may or may not be preempted by another thread following this instruction. 01b = **Atomic**. Prevent any thread switch immediately following this instruction. Always execute the next instruction (which may not be next sequentially if the current instruction branches). Atomic can be used with send, sends, sendc, sendsc ONLY. 10b = **Switch**. Force a switch to another thread after this instruction and before the next instruction. 11b = Reserved
UIP	**Update Instruction Pointer.** A Doubleword Signed Integer offset relative to the current IP (which references the current instruction) in units of bytes. Typically an immediate value in the instruction. For the *brc* (Branch Converging) instruction, both JIP and UIP can be contained in a register.

In the following table of operand fields, use just the last part of the field name. For example, to find the Src1.ChanSel field, look for ChanSel.

Source or Destination Operand Fields (Alphabetically by Short Name) [CHV, BSW]

Field	Description
AddrImm	**Address Immediate.** A 10-bit signed integer offset in units of bytes, only used with the Indirect Addressing Mode. In that addressing mode, the Address Immediate value is added to an address subregister value to determine the operand's address in the GRF. Allowed for any GRF register operand, destination or source. ARF registers cannot be accessed with indirect addressing. The Address Immediate field cannot cover the 4K-byte range of the thread's GRF. Whatever address subregister value is used along with this offset provides a window into the GRF, limited by the offset range. In the Align16 Access Mode, the low four bits of AddrImm are zero and do not appear in the instruction format. Format = S9 Range = [-512, 511]
AddrMode	**Addressing Mode.** Whether the destination register and subregister are encoded in the instruction (Direct Addressing) or calculated using the contents of an address subregister and an offset (Indirect Addressing). This field applies to source and destination register operands, for instructions with 1 or 2 source operands. 0 = Direct Addressing (“Direct”). Direct register addressing. 1 = Indirect Addressing (“Register-Indirect” or “Indirect”). Register-indirect register addressing. Instructions with 3 source operands use Direct Addressing.
AddrSubRegNum	**Address Sub-Register Number.** The address register contains 16 Word-sized subregisters. This field is only used with the Indirect Addressing Mode and specifies the address subregister containing a value added to the Address Immediate value to determine the operand address. Format = U4 Range = [0, 15] Some variations of Indirect Addressing use multiple address subregisters, where AddrSubRegNum determines the first subregister used.
ChanEn	**Channel Enable.** Dst.ChanEn, used only for the destination operand and only in the Align16 Access Mode. Provides four channel enable bits applied modulo four to all *ExecSize* channels. For example, 0xF enables all channels, 0 disables all channels, 0xA enables odd-numbered channels, and so on. The assembler mnemonics are **x**, **y**, **z**, and **w** for channels 0, 1, 2, and 3 respectively. If MaskCtrl is 1 (mnemonic **NoMask**) then all channels are enabled regardless of the ChanEn value, equivalent to ChanEn of 0xF (**xyzw**). Predication and execution masking, in addition to

Field	Description
	ChanEn and MaskCtrl, determine what channels are actually written. For i in 0, 1, 2, 3: Bit i = 0 For channel j where j % 4 == i, disable writing that channel. Bit i = 1 For channel j where j % 4 == i, enable writing that channel.
ChanSel	**Channel Select.** This field controls the channel swizzle for a non-immediate source operand in the Align16 access mode. It is not used for immediate operands, destination operands, or in the Align1 access mode. The normally sequential channel assignment can be altered by explicitly identifying neighboring data elements for each channel. Out of the 8-bit field, 2 bits are assigned for each channel within the group of 4. ChanSel[1:0], [3:2], [5:4] and [7:6] are for channel 0 (**x**), 1 (**y**), 2 (**z**), and 3 (**w**) in the group, respectively. When operating on 64-bit operands, these channel selects must be used in pairs to select a contiguous 64-bit source. For example with an execution size of 8, r0.0<4>.zywz:f assigns the channels as follows: Chan0 = Data2, Chan1 = Data1, Chan2 = Data3, Chan3 = Data2; Chan4 = Data6, Chan5 = Data5, Chan6 = Data7, Chan7 = Data6. The 2-bit Channel Selection field for each channel within the group of 4 is defined as: 00b = **x**. Channel 0 is selected for the corresponding execution channel. 01b = **y**. Channel 1 is selected for the corresponding execution channel. 10b = **z**. Channel 2 is selected for the corresponding execution channel. 11b = **w**. Channel 3 is selected for the corresponding execution channel. Note: When using channel select for 64-bit operands, the valid selects are **.xy** and **.zw**. This is required to pick a pair of DWords.
Desc	**Message descriptor.** A 31 bit immediate message descriptor for *send*, *sendc*, *sends* and *sendsc* instructions.This field is not used for other instructions.
DstType	**Destination Type.** Dst.DstType specifies the numeric data type of the destination operand dst. The bits of the destination operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. For a *send* or *sendc* instruction, this field applies to CurrDst, the current destination operand. Three source instructions use a 3-bit encoding that allows fewer data types. Encoding for one or two source instructions: 0000b = **:ud**. Unsigned Doubleword integer. 0001b = **:d**. Signed Doubleword integer. 0010b = **:uw**. Unsigned Word integer. 0011b = **:w**. Signed Word integer. 0100b = **:ub**. Unsigned Byte integer. 0101b = **:b**. Signed Byte integer.

Field	Description
	0110b = **:df**. Double precision Float (64-bit). 0111b = **:f**. Single precision Float (32-bit). 1000b = **:uq**. Unsigned Quadword integer. 1001b = **:q**. Signed Quadword integer. 1010b = **:hf**. Half Float (16-bit). 1011b to 1111b = Reserved. Encoding for three source instructions: 000b = **:f**. Single precision Float (32-bit). 001b = **:d**. Signed Doubleword integer. 010b = **:ud**. Unsigned Doubleword integer. 011b = **:df**. Double precision Float (64-bit). 100b = **:hf**. Half precision Float (16-bit). 101b - 111b. Reserved.
ExDesc	**Extended message descriptor.** A 32 bit immediate extended message descriptor for *send*, *sendc*, *sends* and *sendsc* instructions.This field is not used for other instructions.
HorzStride	**Horizontal Stride.** Is the distance in units of data element size between two adjacent data elements within a row (horizontal) in the register region for an operand. A horizontal stride of 0 is used for a row that is one-element wide, useful when an instruction repeats a column value or repeats a scalar value. For example, adding a single column to every column in a 2D array or adding a scalar to every element in a 2D array uses HorzStride of 0. A horizontal stride of 1 indicates that elements are adjacent within a row. References to HorzStride in this volume normally reference the value not the encoding, so there are references to HorzStride of 4, which is encoded as 11b. This field applies to both source and destination register operands. This field is used with both direct and indirect addressing. 00b = 0 Elements 01b = 1 Element 10b = 2 Elements 11b = 4 Elements See the Register Region Restrictions section for rules that constrain *HorzStride* in relation to other region parameters.
Imm[28:0]	A 29-bit immediate message descriptor for a *send* or *sendc* instruction. This field is not used for other instructions.

Field	Description
Imm32	A 32-bit immediate data field for an immediate source operand. Only one source operand can be immediate, the last source operand. Of course a destination operand is never immediate. For a two-source instruction, src1 can be immediate; for a one-source instruction src0 can be. The source type for an immediate operand cannot be B or UB (signed or unsigned byte). The source type can be one of the packed vector types that are only allowed as immediate operands: V, UV, or VF. For the W or UW (signed or unsigned word) source types, the 16-bit value must be replicated in both the low and high words of the 32-bit immediate value. The low order bits are directly used when fewer than 32-bits are needed for the source type. The 32-bit value is not coerced into the designated type. See the Numeric Data Types section for information about data types and their ranges.
Imm64	A 64-bit immediate data field for an immediate source operand, only used for a one-source instruction with a 64-bit Source Type (DF, Q, or UQ).
RegFile	**Register File.** Select a source or destination register file or indicate an immediate source operand: 00b = ARF Architecture Register File. Only allowed for src0 or destination. 01b = GRF General Register File. Allowed for any source or destination. 10b = Reserved. 11b = Immediate operand. Only allowed for the last source operand. Not allowed for the destination operand or for any other source operand. Note that for flow control instructions requiring two offsets, regfile of source0 is required to be immediate since the 64b for immediates occupy the DW2 and DW3.
RegNum	**Register Number.** The register number for the operand. For a GRF register, is the part of a register address that aligns to a 256-bit (32-byte) boundary. For an ARF register, this field is encoded such that MSBs identify the architecture register type and LSBs provide the register number. An ARF register can only be dst or src0. Any src1 or src2 operands cannot be ARF registers. RegNum and SubRegNum together provide the byte-aligned address for the origin of a register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands. Format = U8, if RegFile = GRF. 0x00 to 0x7F = Register number in the range [0, 127]. 0x80 to 0xFF = Reserved.

Field	Description
	Format = 8-bit encoding, if RegFile = ARF. This field encodes the architecture register type as well as providing the register number. See the ARF Registers Overview section and the sections for individual ARF registers for details.
RepCtrl	**Replicate Control.** This field is only present in three-source instructions, for each of the three source operands. It controls replication of the starting channel to all channels in the execution size. ChanSel does not apply when Replicate Control is set. This is applicable to 32b datatypes and 16b datatype. 64b datatypes cannot use the replicate control. 0 = No replication. 1 = Replicate across all channels.
SelReg32Desc	**Select Reg32 for message descriptor.** In *sends* or *sendsc* instruction, refers to the selection of Reg32 for the message descriptor field. 0 = Desc. 1 = Reg32. First Dword of Address Register is used for message descriptor.
SrcMod	**Source Modifier.** Specify any numeric (normally) or logical (for logic instructions) modification to a source value before delivery to the execution pipe. The numeric value of each data element of a source operand can optionally have its absolute value taken, its sign inverted (arithmetic negation), or both (absolute value followed by arithmetic negation producing a guaranteed negative value). When used with logic instructions (*and*, *not*, *or*, *xor*), this field indicates whether the source bits are inverted (bitwise NOT) before delivery to the execution pipe, regardless of the source type. This field only applies to source operands. It does not apply to the destination. This field is not present for an immediate source operand. For no modification, there is no assembler notation or syntax. Encoding for all instructions other than logic instructions: 00b = No modification (normal). 01b = **(abs)**. Absolute value. 10b = **–**. Negation. 11b = **–(abs)**. Negation of the absolute value (forced negative value). Encoding for logic instructions: 00b = No modification (normal). 01b = No modification (normal). This encoding cannot be selected in the assembler syntax. 10b = **–** Indicates a bitwise NOT, inverting the source bits. 11b = No modification (normal). This encoding cannot be selected in the assembler syntax.

Field	Description
SrcType	**Source Type.** Specifies the numeric data type of a source operand. In a two-source instruction, each source operand has its own source type field. In a three-source instruction, one source type is used for all three source operands. The bits of a source operand are interpreted as the identified numeric data type, rather than coerced into a type implied by the operator. Depending on the RegFile field for the source, this field uses one of two encodings. For a non-immediate source (from a register file), use the Source Register Type Encoding, which is identical to the Destination Type encoding. For an immediate source, use the Source Immediate Type Encoding, which does not support signed or unsigned byte immediate values and does support the three packed vector types, V, UV, and VF. Note that three-source instructions do not support immediate operands, that only the second source (src1) of a two-source instruction can be immediate, and that 64-bit immediate values (DF, Q, or UQ) can only be used with one-source instructions. In a two-source instruction with a V (Packed Signed Half-Byte Integer Vector) or UV (Packed Unsigned Half-Byte Integer Vector) immediate operand, the other source operand must have a type compatible with packed word execution mode, one of B, UB, W, or UW. Note that DF (Double Float) and HF (Half Float) have different encodings in the Source Regster Type Encoding and the Source Immediate Type Encoding. The Source Register Type Encoding and Source Immediate Type Encoding lists apply to instructions with one or two source operands. Source Register Type Encoding: 0000b = UD. Unsigned Doubleword integer. 0001b = D. Signed Doubleword integer. 0010b = UW. Unsigned Word integer. 0011b = W. Signed Word integer. 0100b = UB. Unsigned Byte integer. 0101b = B. Signed Byte integer. 0110b = DF. Double precision Float (64-bit). 0111b = F. Single precision Float (32-bit). 1000b = UQ. Unsigned Quadword integer. 1001b = Q. Signed Quadword integer. 1010b = HF. Half Float (16-bit). 1011b to 1111b = Reserved. Source Immediate Type Encoding: 0000b = UD. Unsigned Doubleword integer. 0001b = D. Signed Doubleword integer.

Field	Description
	0010b = UW. Unsigned Word integer. 0011b = W. Signed Word integer. 0100b = UV. Packed Unsigned Half-Byte Integer Vector, 8 x 4-Bit Unsigned Integer. 0101b = VF. Packed Restricted Float Vector, 4 x 8-Bit Restricted Precision Floating-Point Number. 0110b = V. Packed Signed Half-Byte Integer Vector, 8 x 4-Bit Signed Integer. 0111b = F. Single precision Float (32-bit). 1000b = UQ. Unsigned Quadword integer. 1001b = Q. Signed Quadword integer. 1010b = DF. Double precision Float (64-bit). 1011b = HF. Half Float (16-bit). 1100b to 1111b = Reserved. Three source instructions use one SrcType field for all source operands, with a 3-bit encoding that allows fewer data types: Encoding for three source instructions: 000b = **:f**. Single precision Float (32-bit). 001b = **:d**. Signed Doubleword integer. 010b = **:ud**. Unsigned Doubleword integer. 011b = **:df**. Double precision Float (64-bit). 100b = **:hf**. Half precision Float (16-bit). 101b - 111b. Reserved. Three source instructions can use operands with mixed-mode precision. When SrcType field is set to :f or :hf it defines precision for source 0 only, and fields Src1Type and Src2Type define precision for other source operands: 0b = **:f**. Single precision Float (32-bit). 1b = **:hf**. Half precision Float (16-bit).
SubRegNum	**Sub-Register Number.** The subregister number for the operand. For a GRF register, is the byte address within a 256-bit (32-byte) register. For an ARF register, determines the sub-register number according to the specified encoding for the given architecture register. RegNum and SubRegNum together provide the byte-aligned address for the origin of a GRF register region. RegNum provides bits 12:5 of that address. For one-source and two-source instructions, SubregNum provides bits 4:0. For three-source instructions, the address must be DWord-aligned; SubRegNum provides bits 4:2 of the address and bits 1:0 are zero. **Note:** The recommended instruction syntax uses subregister numbers within the GRF in units

Field	Description
	of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size. This field is present for the direct addressing mode and not present for indirect addressing. This field applies to both source and destination operands. Format = U5, if RegFile = GRF and the instruction has fewer than three source operands. 0x00 to 0x1F = Sub-Register number in the range of [0, 31]. Format = U3, if RegFile = GRF and the instruction has three source operands. 0x0 to 0x7 = Sub-Register number MSBs in the range of [0,7]. The two LSBs are zero. Format = U4, if RegFile = GRF and the instruction has three source operands. 0x0 to 0x15 = Sub-Register number MSBs in the range of [0,15]. Format = 5-bit encoding, if RegFile = ARF. See the ARF Registers Overview section and the sections for individual ARF registers for details.
VertStride	**Vertical Stride.** The vertical stride of a source operand's register region in units of data element size. Supported values are 0, powers of 2 from 1 to 32, and a special encoding used for indirect addressing in Align1 mode. Values greater than 32 are not supported due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). The special encoding 1111b (0xF) is only valid when the operand is in register-indirect addressing mode (AddrMode = 1). If this field is set to 0xF, one or more sub-registers of the address registers may be used to compute source addresses. Each address sub-register provides the origin for a row of data elements. The number of address sub-registers used is equal to instruction's ExecSize / source operand's Width. This field only applies to source operands. It does not apply to the destination. For Align16 access mode, only encodings of 0000b, 0010b, and 0011b are allowed. Other codes are reserved. Note 1: A Vertical Stride larger than 32 is not allowed due to the restriction that a source operand must reside within two adjacent 256-bit registers (64 bytes total). Note 2: In Align16 access mode, as encoding 1111b is reserved, only single-index indirect addressing is supported. Note 3: If indirect addressing is supported for src1, the encoding 1111b is reserved for src1 and only single-index indirect addressing is supported.

Field	Description
	Note 4: The encoding 0010b is used for 64-bit operands (types DF, Q, or UQ). 0000b = 0 Elements 0001b = 1 Element. Align1 mode only. 0010b = 2 Elements 0011b = 4 Elements 0100b = 8 Elements. Align1 mode only. 0101b = 16 Elements (applies to byte or word operands only). Align1 mode only. 0110b = 32 Elements (applies to byte operands only). Align1 mode only. 0111b-1110b = Reserved. 1111b = VxH or Vx1 mode (only valid for register-indirect addressing in Align1 mode).
Width	**Width.** The number of elements in the horizontal dimension of the region for a source operand. This field cannot exceed the ExecSize field of the instruction. This field only applies to source operands. It does not apply to the destination. 000b = 1 Element 001b = 2 Elements 010b = 4 Elements 011b = 8 Elements 100b = 16 Elements 101b-111b = Reserved Note that with ExecSize of 32, because the maximum Width is 16, there are at least two rows in a source region.

EU Compact Instructions

On receiving an instruction with bit 29 (CmptCtrl) set, HW recognizes it as a 64-bit compact instruction. Hardware then uses the index fields inside the compact instruction to lookup values in the associated compaction tables, then uses the table outputs along with other fields in the compact instruction to reconstruct the 128-bit native-sized instruction.

Description
All flow control instructions use the new offset format, a signed 32-bit offset in units of bytes.

The native 128-bit instruction format provides access to all instruction options. Only some instruction options and combinations of instruction options can be represented in the compact instruction formats.

Which native instructions can be represented as compact instructions and the details of the compact instruction formats and the compaction tables used may change with each processor generation.

In the following instruction format tables the Mapping Bits and Mapping Description columns describe the mappings into native instruction fields.

EU Compact Instruction Format

The following table describes the EU compact instruction format for CHV. CHV adds support for compacting instructions with three source operands.

CHV, BSW supports instruction compaction for 3-source instructions as well as for 1 or 2-source instructions. The next two tables provide the CHV compact instruction formats for 1 or 2-source instructions first and then for 3-source instructions.

The CHV compact instruction format for 1 or 2-source instructions is essentially identical to the compact instruction format for earlier generations, but the compact fields expand to somewhat different fields in the native instruction format, as the native instruction format changed for CHV.

GEN Compact Instruction Format CHV 1 or 2 Source Operands

Bits	Size	Mapping Bits	Compact Name	Mapping Description
63:56	8	108:101 (Not Imm.) or 103:96 (Imm.)	Src1.RegNum	Src1.RegNum if not immediate. Imm32[7:0] if immediate.
55:48	8	76:69	Src0.RegNum	Src0.RegNum.
47:40	8	60:53	Dst.RegNum	Dst.RegNum.
39:35	5	120:109 (Not Imm.) or 127:104	Src1Index	If not an immediate operand, lookup one of 32 12-bit values that maps to bits 120:109. That value is used (from MSB to LSB) for the Src1.VertStride, various Src1 bit fields

Bits	Size	Mapping Bits	Compact Name	Mapping Description
		(Imm.)		based on AccessMode (Src1.ChanSel[7:4], Src1.Width, Src1.HorzStride), Src1.AddrMode, and Src1.SrcMod bit fields. If an immediate operand, there is no lookup. Determines bits 127:104 (Imm32[31:8]) as follows: map bits 39:35 directly to bits 108:104. Sign extend to fill bits 127:109. Compact format bit 39 is thus copied to all of bits 127:108 for an immediate operand.
34:30	5	88:77	Src0Index	Lookup one of 32 12-bit values. That value is used (from MSB to LSB) for the Src0.VertStride, various Src0 bit fields based on AccessMode (Src0.ChanSel[7:4], Src0.Width, Src0.HorzStride), Src0.AddrMode, and Src0.SrcMod bit fields. Note that this field spans a DWord boundary within the QWord compacted instruction.
29	1	29	CmptCtrl	Compaction Control. The same in both the compact and native formats: 0: Regular instruction, not compacted. 1: Compacted instruction.
28	1	Not mapped.	Reserved	Not mapped.
27:24	4	27:24	CondModifier	CondModifier. The same in both the compact and native formats.
23	1	28	AccWrCtrl	AccWrCtrl.
22:18	5	100:96, 68:64, 52:48	SubRegIndex	Lookup one of 32 15-bit values. That value is used (from MSB to LSB) for various fields for Src1, Src0, and Dst, including ChanEn/ChanSel, SubRegNum, and AddrImm[4] or AddrImm[4:0], depending on AddrMode and AccessMode.
17:13	5	63:61, 94:89, 46:35	DataTypeIndex	Lookup one of 32 21-bit values. That value is used (from MSB to LSB) for the Dst.AddrMode, Dst.HorzStride, Src1.SrcType, Src1.RegFile, Src0.SrcType, Src0.RegFile, Dst.DstType, and Dst.RegFile bit fields.
12:8	5	33:32, 31, 23:12, 10:9, 34, 8	ControlIndex	Lookup one of 32 19-bit values. That value is used (from MSB to LSB) for the FlagRegNum/FlagSubRegNum, Saturate, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, DepCtrl, MaskCtrl, and AccessMode bit fields.
6:0	7	6:0	Opcode	Opcode. The same in both the compact and native formats.

GEN Compact Instruction Format 3 Source Operands

Bits	Size	Mapping Bits	Compact Name	Mapping Description
63:57	7	124:118	Src2.RegNum[6:0]	Src2.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src2.RegNum[7].
56:50	7	103:97	Src1.RegNum[6:0]	Src1.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src1.RegNum[7].
49:43	7	82:76	Src0.RegNum[6:0]	Src0.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src0.RegNum[7].
42:40	3	117:115	Src2.SubRegNum	Src2.SubRegNum.
39:37	3	96:94	Src1.SubRegNum	Src1.SubRegNum.
36:34	3	75:73	Src0.SubRegNum	Src0.SubRegNum.
33	1	106	Src2.RepCtrl	Src2.RepCtrl.
32	1	85	Src1.RepCtrl	Src1.RepCtrl.
31	1	31	Saturate	Saturate.
29	1	29	CmptCtrl	Compaction Control. The same in both the compact and native formats: 0: Regular instruction, not compacted. 1: Compacted instruction.
28	1	64	Src0.RepCtrl	Src0.RepCtrl.
27:19	9	Not mapped	Reserved	Not mapped.
18:12	7	63:56	Dst.RegNum[6:0]	Dst.RegNum[7:0] with MSB of zero and [6:0] from the compact instruction.
11:10	2	125, 104, 83, 114:107, 93:86, 72:65, 55:49, 48:43,	SourceIndex	Lookup one of four 46-bit values. That value is used (from MSB to LSB) for the Src2.RegNum[7], Src1.RegNum[7], Src0.RegNum[7], Src2.ChanSel, Src1.ChanSel, Src0.ChanSel, Dst.SubRegNum, Dst.ChanEnable, Dst.DstType,

		42:37		SrcType, Src2.Modifier, Src1.Modifier, and Src0.Modifier bit fields.
9:8	2	34, 33:32, 28:8	ControlIndex	Lookup one of four 24-bit values. That value is used (from MSB to LSB) for the MaskCtrl, FlagRegNum/FlagSubRegNum, AccWrCtrl, CondModifier, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, NibCtrl, DepCtrl, and AccessMode bit fields.
7	1	Not mapped	Reserved	Not mapped.
6:0	7	6:0	Opcode	Opcode. The same in both the compact and native formats.

GEN Compact Instruction Format 3 Source Operands

Bits	Size	Mapping Bits	Compact Name	Mapping Description
63:57	7	124:118	Src2.RegNum[6:0]	Src2.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src2.RegNum[7].
56:50	7	103:97	Src1.RegNum[6:0]	Src1.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src1.RegNum[7].
49:43	7	82:76	Src0.RegNum[6:0]	Src0.RegNum[6:0]. The SourceIndex field in the compact instruction determines Src0.RegNum[7].
42:40	3	117:115	Src2.SubRegNum	Src2.SubRegNum.
39:37	3	96:94	Src1.SubRegNum	Src1.SubRegNum.
36:34	3	75:73	Src0.SubRegNum	Src0.SubRegNum.
33	1	106	Src2.RepCtrl	Src2.RepCtrl.
32	1	85	Src1.RepCtrl	Src1.RepCtrl.
31	1	31	Saturate	Saturate.
29	1	29	CmptCtrl	Compaction Control. The same in both the compact and native formats: 0: Regular instruction, not compacted. 1: Compacted instruction.
28	1	64	Src0.RepCtrl	Src0.RepCtrl.
27:19	9	Not mapped	Reserved	Not mapped.
18:12	7	63:56	Dst.RegNum[6:0]	Dst.RegNum[7:0] with MSB of zero and [6:0] from the compact instruction.
11:10	2	126, 125, 105, 104, 84, 83, 114:107, 93:86, 72:65, 55:49, 48:43, 42:37	SourceIndex	Lookup one of four 49-bit values. That value is used (from MSB to LSB) for the Src2.SubRegNum, Src2.RegNum[7], Src1.RegNum, Src1.RegNum[7], Src0.SubRegNum, Src0.RegNum[7], Src2.ChanSel, Src1.ChanSel, Src0.ChanSel, Dst.SubRegNum, Dst.ChanEnable, Dst.DstType, SrcType, Src2.Modifier, Src1.Modifier, and Src0.Modifier bit fields.
9:8	2	36:35, 34, 33:32, 28:8	ControlIndex	Lookup one of four 26-bit values. That value is used (from MSB to LSB) for the Src1.Type, Src2.Type MaskCtrl, FlagRegNum/FlagSubRegNum, AccWrCtrl, CondModifier, ExecSize, PredInv, PredCtrl, ThreadCtrl, QtrCtrl, NibCtrl, DepCtrl, and AccessMode bit fields.
7	1	Not mapped	Reserved	Not mapped.
6:0	7	6:0	Opcode	Opcode. The same in both the compact and native formats.

The following diagrams are an alternate presentation of the CHV compact instruction formats.

GEN Compact Instruction Format [CHV, BSW] 1 or 2 Source Operands

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	Description	Bits in 128bits Format	Description (Imm. Src0 or Src1)	Bits in 128bits Format (Imm. Src0 or Src1)
	8	63	56	8	Src1 RegNum	[108:101]	Imm[23:16] Imm[7:0]	[119:112] [103:96]
	8	55	48	8	Src0 RegNum	[76:69]	Src0 RegNum	[76:69]
	8	47	40	8	Dst RegNum	[60:53]	Dst RegNum	[60:53]
	5	39	35	5	Src1Index[4:0]	[120:109]	Src1Index[4:0]	[127:120] [111:104]
1	3	34	32	3				
	2	31	30	2	Src0Index[4:0]	[88:77]	Src0Index[4:0]	[88:77]
	1	29	29	1	CmptCtrl	[29]	CmptCtrl	[29]
	1	28	28	1	Reserved		Reserved	
	4	27	24	4	CondModifier	[27:24]	CondModifier	[27:24]
	1	23	23	1	AccWrCtrl	[28]	AccWrCtrl	[28]
	5	22	18	5	SubRegIndex[4:0]	[100:96] [68:64] [52:48]	SubRegIndex[4:0]	[100:96] [68:64] [52:48]
	5	17	13	5	DataTypeIndex[4:0]	[63:61] [94:89][46:35]	DataTypeIndex[4:0]	[63:61] [94:89][46:35]
	5	12	8	5	ControlIndex[4:0]	[33:32] [31] [23:12][10:9][34][8]	ControlIndex[4:0]	[33:32] [31] [23:12][10:9][34][8]
	1	7	7	1	DebugCtrl	[30]	DebugCtrl	[30]
0	7	6	0	7	Opcode	[6:0]	Opcode	[6:0]

GEN Compact Instruction Format 3 Source Operands

DW #	Instr Bits Alloc	High Bit	Low Bit	Instr Bits Used	Description	Bits in 128bits Format
	7	63	57	7	*Src2 RegNum*	*[124:118]*
	7	56	50	7	*Src1 RegNum*	*[103:97]*
	7	49	43	7	*Src0 RegNum*	*[82:76]*
	3	42	40	3	*Src2 SubRegNum*	*[117:115]*
	3	39	37	3	*Src1 SubRegNum*	*[96:94]*
	3	36	34	3	*Src0 SubRegNum*	*[75:73]*
	1	33	33	1	*Src2 Replicate*	*[106]*
1	1	32	32	1	*Src1 Replicate*	*[85]*
	1	31	31	1	*Saturate*	*[31]*
	1	30	30	1	*DebugCtrl*	*[30]*
	1	29	29	1	*CmptCtrl*	*[29]*
	1	28	28	1	*Src0 Replicate*	*[64]*
	9	27	19	9	*Reserved*	
	7	18	12	7	*Dst RegNum*	*[63:56]*
	2	11	10	2	*SourceIndex[1:0]*	*[125][104][83][114:107][93:86][72:65][55:49][46:43][**42:37**]*
	2	9	8	2	*ControlIndex[1:0]*	*[**34**][33:32][28][27:24][**23:21**][20:14][**13:12**][11][10:8]*
	1	7	7	1	*Reserved*	
0	7	6	0	7	*Opcode*	*[6:0]*

EU Instruction Compaction Tables

The following four tables describe the mappings for the ControlIndex, DataTypeIndex, SubRegIndex, Src0Index, and Src1Index fields in the 1 or 2 source operand compact instruction format.

ControlIndex Compact Instruction Field Mappings 1 or 2 Source Operands

ControlIndex	19-Bit Mapping	Mapped Meaning
0	0000000000000000010	Align1 | We | (1) | f0.0
1	0000100000000000000	Align1 | (4) | f0.0
2	0000100000000000001	Align16 | (4) | f0.0
3	0000100000000000010	Align1 | We | (4) | f0.0
4	0000100000000000011	Align16 | We | (4) | f0.0
5	0000100000000000100	Align1 | NoDDClr | (4) | f0.0
6	0000100000000000101	Align16 | NoDDClr | (4) | f0.0

ControlIndex	19-Bit Mapping	Mapped Meaning
7	0000100000000000111	Align16 | We | NoDDClr | (4) | f0.0
9	0000100000000001001	
10	0000100000000001101	Align16 | NoDDClr | (4) | f0.0
11	0000110000000000000	Align1 | Q1 | (8) | f0.0
12	0000110000000000001	Align16 | Q1 | (8) | f0.0
13	0000110000000000010	Align1 | We | Q1 | (8) | f0.0
14	0000110000000000011	Align16 | We | Q1 | (8) | f0.0
15	0000110000000000100	Align1 | NoDDClr | Q1 | (8) | f0.0
16	0000110000000000101	Align16 | NoDDClr | Q1 | (8) | f0.0
17	0000110000000000111	Align16 | We | NoDDClr | Q1 | (8) | f0.0
18	0000110000000001001	Resreved
19	0000110000000001101	Align16 | NoDDClr | Q1 | (8) | f0.0
20	0000110000000010000	Align1 | Q2 | (8) | f0.0
21	0000110000100000000	Align1 | Q1 | +f.xyzw | (8) | f0.0
22	0001000000000000000	Align1 | H1 | (16) | f0.0
23	0001000000000000010	Align1 | We | H1 | (16) | f0.0
24	0001000000000000100	Align1 | NoDDClr | H1 | (16) | f0.0
25	0001000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.0
26	0010110000000000000	Align1 | Q1 | (8) | .sat | f0.0
27	0010110000000010000	Align1 | Q2 | (8) | .sat | f0.0
28	0011000000000000000	Align1 | H1 | (16) | .sat | f0.0
29	0011000000100000000	Align1 | H1 | +f.xyzw | (16) | .sat | f0.0
30	0101000000000000000	Align1 | H1 | (16) | f0.1
31	0101000000100000000	Align1 | H1 | +f.xyzw | (16) | f0.1

DataTypeIndex Compact Instruction Field Mappings 1 or 2 Source Operands

DataTypeIndex	21-Bit Mapping	Mapped Meaning
0	001000000000000000001	r:ud | a:ud | a:ud | <1> | dir |
1	001000000000001000000	a:ud | r:ud | a:ud | <1> | dir |
2	001000000000001000001	r:ud | r:ud | a:ud | <1> | dir |
3	001000000000011000001	r:ud | i:ud | a:ud | <1> | dir |
4	001000000000101011101	r:f | r:d | a:ud | <1> | dir |
5	001000000010111011101	r:f | i:vf | a:ud | <1> | dir |
6	001000000011101000001	r:ud | r:f | a:ud | <1> | dir |
7	001000000011101000101	r:d | r:f | a:ud | <1> | dir |

DataTypeIndex	21-Bit Mapping	Mapped Meaning
8	001000000011101011101	r:f | r:f | a:ud | <1> | dir |
9	001000001000001000001	r:ud | r:ud | r:ud | <1> | dir |
10	001000011000001000000	a:ud | r:ud | i:ud | <1> | dir |
11	001000011000001000001	r:ud | r:ud | i:ud | <1> | dir |
12	001000101000101000101	r:d | r:d | r:d | <1> | dir |
13	001000111000101000100	a:d | r:d | i:d | <1> | dir |
14	001000111000101000101	r:d | r:d | i:d | <1> | dir |
15	001011100011101011101	r:f | r:f | a:f | <1> | dir |
16	001011101011100011101	r:f | a:f | r:f | <1> | dir |
17	001011101011101011100	a:f | r:f | r:f | <1> | dir |
18	001011101011101011101	r:f | r:f | r:f | <1> | dir |
19	001011111011101011100	a:f | r:f | i:f | <1> | dir |
20	000000000010000001100	a:w | a:ub | a:ud | <0> | dir |
21	001000000000001011101	r:f | r:ud | a:ud | <1> | dir |
22	001000000000101000101	r:d | r:d | a:ud | <1> | dir |
23	001000001000001000000	a:ud | r:ud | r:ud | <1> | dir |
24	001000101000101000100	a:d | r:d | r:d | <1> | dir |
25	001000111000100000100	a:d | a:d | i:d | <1> | dir |
26	001001001001000001001	r:uw | a:uw | r:uw | <1> | dir |
27	001010111011101011101	r:f | r:f | i:vf | <1> | dir |
28	001011111011101011101	r:f | r:f | i:f | <1> | dir |
29	001001111001101001100	a:w | r:w | i:w | <1> | dir |
30	001001001001001001000	a:uw | r:uw | r:uw | <1> | dir |
31	001001011001001001000	a:uw | r:uw | i:uw | <1> | dir |

SubRegIndex Compact Instruction Field Mappings 1 or 2 Source Operands

SubRegIndex	15-Bit Mapping	Mapped Meaning
0	000000000000000	0 | 0 | 0 |
1	000000000000001	0.x | 0.xx | 0.xx
2	000000000001000	8 | 0 | 0 |
3	000000000001111	0.xyzw | 0.xx | 0.xx
4	000000000010000	16 | 0 | 0 |
5	000000010000000	0 | 4 | 0 |
6	000000100000000	0 | 8 | 0 |
7	000000110000000	0 | 12 | 0 |
8	000001000000000	0 | 16 | 0 |
9	000001000010000	16 | 16 | 0 |
10	000001010000000	0 | 20 | 0 |
11	001000000000000	0 | 0 | 4 |
12	001000000000001	0.x | 0.xx | 0.xy
13	001000010000001	0.x | 0.xy | 0.xy
14	001000010000010	0.y | 0.xy | 0.xy
15	001000010000011	0.xy | 0.xy | 0.xy
16	001000010000100	0.z | 0.xy | 0.xy
17	001000010000111	0.xyz | 0.xy | 0.xy
18	001000010001000	0.w | 0.xy | 0.xy
19	001000010001110	0.yzw | 0.xy | 0.xy
20	001000010001111	0.xyzw | 0.xy | 0.xy
21	001000110000000	0 | 12 | 4 |
22	001000111101000	0.w | 0.ww | 0.xy
23	010000000000000	0 | 0 | 8 |
24	010000110000000	0 | 12 | 8 |
25	011000000000000	0 | 0 | 12 |
26	011110010000111	0.xyz | 0.xy | 0.ww
27	100000000000000	0 | 0 | 16 |
28	101000000000000	0 | 0 | 20 |
29	110000000000000	0 | 0 | 24 |
30	111000000000000	0 | 0 | 28 |
31	111000000011100	28 | 0 | 28 |

Src0Index or Src1Index Compact Instruction Field Mappings 1 or 2 Source Operands

Src0Index or Src1Index	12-Bit Mapping	Mapped Meaning
0	000000000000	dir | <0;1,0>
1	000000000010	(-) | dir | <0;1,0>
2	000000010000	dir | <0;>.zx
3	000000010010	(-) | dir | <0;>.zx
4	000000011000	dir | <0;>.wx
5	000000100000	dir | <0;>.xy
6	000000101000	dir | <0;>.yy
7	000001001000	dir | <0;4,1>
8	000001010000	dir | <0;>.zz
9	000001110000	dir | <0;>.zw
10	000001111000	dir | <0;8,4> / dir | <0;>.ww
11	001100000000	dir | <4;>.xx
12	001100000010	(-) | dir | <4;>.xx
13	001100001000	dir | <4;>.yx
14	001100010000	dir | <4;>.zx
15	001100010010	(-) | dir | <4;>.zx
16	001100100000	dir | <4;>.xy
17	001100101000	dir | <4;>.yy
18	001100111000	dir | <4;>.wy
19	001101000000	dir | <4;4,0>
20	001101000010	(-) | dir | <4;4,0>
21	001101001000	dir | <4;>.yz
22	001101010000	dir | <4;>.zz
23	001101100000	dir | <4;>.xw
24	001101101000	dir | <4;>.yw
25	001101110000	dir | <4;>.zw
26	001101110001	(abs) | dir | <4;>.zw
27	001101111000	dir | <4;>.ww
28	010001101000	dir | <8;8,1>
29	010001101001	(abs) | dir | <8;8,1>
30	010001101010	(-) | dir | <8;8,1>
31	010110001000	dir | <16;16,1>

The following tables describe the mappings for the ControlIndex and SourceIndex fields in the 3 source operand compact instruction format.

ControlIndex Compact Instruction Field Mappings 3 Source Operands

ControlIndex	24-Bit Mapping	Mapped Meaning
0	100000000110000000000001	(8) Q1 NoMask Align16
1	000000000110000000000001	(8) Q1 Align16
2	000000001000000000000001	(16) H1 Align16
3	000000001000000000100001	(16) H2 Align16

ControlIndex Compact Instruction Field Mappings 3 Source Operands

ControlIndex	26-Bit Mapping	Mapped Meaning
0	00100000000110000000000001	(8) Q1 NoMask Align16
1	00000000000110000000000001	(8) Q1 Align16
2	00000000001000000000000001	(16) H1 Align16
3	00000000001000000000100001	(16) H2 Align16

SourceIndex Compact Instruction Field Mappings 3 Source Operands

SourceIndex	46-Bit Mapping	Mapped Meaning
0	00011100100111001001110010000011110000000000000	No Negation
1	00011100100111001001110010000011110000000000010	Negate Src0
2	00011100100111001001110010000011110000000001000	Negate Src1
3	00011100100111001001110010000011110000000100000	Negate Src2

SourceIndex Compact Instruction Field Mappings 3 Source Operands

SourceIndex	49-Bit Mapping	Mapped Meaning
0	0000001110010011100100111001000001111000000000000	No Negation
1	0000001110010011100100111001000001111000000000010	Negate Src0
2	0000001110010011100100111001000001111000000001000	Negate Src1
3	0000001110010011100100111001000001111000000100000	Negate Src2

Opcode Encoding

Byte 0 of the 128-bit instruction word contains the opcode. The opcode uses 7 bits. Bit location 7 in byte 0 is reserved for future opcode extension.

The opcodes are encoded and organized into five groups based on the type of operations: Special instructions, move/logic instructions (opcode=00xxxxxb), flow control instructions (opcode=010xxxxb), miscellaneous instructions (opcode=011xxxxb), parallel arithmetic instructions (opcode=100xxxxb), and vector arithmetic instructions (opcode=101xxxxb). Opcodes 110xxxb are reserved.

Note: Opcodes appear in the overall Instruction Set Summary Table as well. The following subsections still serve the purpose of describing various instruction groups.

Move and Logic Instructions

This instruction group has an opcode format of 00xxxxxb.

- The opcodes for move instructions (*mov*, *sel* and *movi*) share the common 5 MSBs in the form of 00000xxb.
- The opcodes for logic instructions (*not*, *and*, *or*, and *xor*) share the common 5 MSBs in the form of 00001xxb.
- The opcodes for shift instructions (*shr, shl,* and *asr*) share the common 4 MSBs in the form of 0001xxxb. Bit 2 indicates arithmetic or logic shift (0 = logic, 1 = arithmic). Bit 1 is always 0 (which is reserved for future extension to support rotation shift as 0 = shift, 1 = rotate). Bit 0 indicates the shift direction (0 = right, 1 = left).
- The opcodes for compare instructions (*cmp* and *cmpn*) share the common 6 MSBs in the form of 001000xb. Bit 0 indicates whether it is a normal compare, *cmp*, or a special compare-NaN, *cmpn*.

Move and Logic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
1	0x01	mov	Component-wise move	1	1
2	0x02	sel	Component-wise selective move based on predication	2	1
3	0x03	movi	Fast component-wise indexed move	1	1
4	0x04	not	Component-wise one's complement (bitwise not)	1	1
5	0x05	and	Component-wise logical AND (bitwise and)	2	1
6	0x06	or	Component-wise logical OR (bitwise or)	2	1
7	0x07	xor	Component-wise logical XOR (bitwise xor)	2	1
8	0x08	shr	Component-wise logical shift right	2	1
9	0x09	shl	Component-wise logical shift left	2	1
10	0x0A	smov	Scattered Move	1	1
11	0x0B	*Reserved*			

Opcode		Instruction	Description	#src	#dst
dec	hex				
12	0x0C	asr	Component-wise arithmetic shift right	2	1
13	0x0D	*Reserved*			
14	0x0E	*Reserved*			
15	0x0F	*Reserved*			
16	0x10	cmp	Component-wise compare, store condition code in destination	2	1
17	0x11	cmpn	Component-wise compare-NaN, store condition code in destination	2	1
18	0x12	csel	Component-wise selective move based on result of compare	3	1
19	0x13	*Reserved*			
20	0x14	*Reserved*			
21	0x15	*Reserved*			
22	0x16	*Reserved*			
23	0x17	bfrev	Reverse bits	1	1
24	0x18	bfe	Bitfield extract	3	1
25	0x19	bfi1	Bitfield insert macro instruction 1, generate mask	2	1
26	0x1A	bfi2	Bitfield insert macro instruction 2, insert based on mask	3	1
27-31	0x1B-0x1F	*Reserved*			

Flow Control Instructions

This instruction group has an opcode format of 010xxxxb.

Flow Control Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
32	0x20	jmpi	Jump indexed	1	0
33	0x21	brd	Branch - Diverging	1	0
34	0x22	if	If	0/2	0
35	0x23	brc	Branch - Converging	1	-
36	0x24	else	Else	1	0
37	0x25	endif	End if	0	0
38	0x26	*Reserved*			
39	0x27	*Reserved*			

Opcode		Instruction	Description	#src	#dst
dec	hex				
40	0x28	break	Break	1	0
41	0x29	cont	Continue	1	0
42	0x2A	halt	Halt	1	0
43	0x2B	calla	Subroutine call absolute	1	1
44	0x2C	call	Subroutine call	1	1
45	0x2D	return	Subroutine return	1	1
46	0x2E	goto	Goto	2	0
47	0x2F	join	Join	2	0

Miscellaneous Instructions

This instruction group has an opcode format of 011xxxxb.

Miscellaneous Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
48	0x30	wait	Wait for (external) notification	1	0
49	0x31	send	Send	1	1
50	0x32	sendc	Conditional Send (based on TDR)	1	1
51	0x33	*Reserved*			
52	0x34	*Reserved*			
53-55	0x35-0x37	*Reserved*			
56	0x38	math	Math functions for extended math pipeline	1/2	1/2
57-63	0x39-0x3F	*Reserved*			

Parallel Arithmetic Instructions

This instruction group has an opcode format of 100xxxxb.

Parallel Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
64	0x40	add	Component-wise addition	2	1
65	0x41	mul	Component-wise multiply	2	1
66	0x42	avg	Component-wise average of the two source operands	2	1
67	0x43	frc	Component-wise floating point truncate-to-minus-infinity fraction	1	1

Opcode		Instruction	Description	#src	#dst
dec	hex				
68	0x44	rndu	Component-wise floating point rounding up (ceiling)	1	1
69	0x45	rndd	Component-wise floating point rounding down (floor)	1	1
70	0x46	rnde	Component-wise floating point rounding toward nearest even	1	1
71	0x47	rndz	Component-wise floating point rounding toward zero	1	1
72	0x48	mac	Component-wise multiply accumulate	2	1
73	0x49	mach	multiply accumulate high	2	1
74	0x4A	lzd	leading zero detection	1	1
75	0x4B	fbh	Find first 1 for UD from msb side, or first 1/0 for D.	1	1
76	0x4C	fbl	First first 1 for UD from lsb side	1	1
77	0x4D	cbit	Count bits set	1	1
78	0x4E	addc	Integer add with carry	2	1 + acc.
79	0x4F	subb	integer subtract with borrow	2	1 + acc.

Vector Arithmetic Instructions

This instruction group has an opcode format of 101xxxxb.

Vector Arithmetic Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
80	0x50	sad2	2-wide sum of absolute difference	2	1
81	0x51	sada2	2-wide sad accumulate	2	1
82-83	0x52-0x53	*Reserved*			
84	0x54	dp4	4-wide dot product for 4-vector	2	1
85	0x55	dph	4-wide homogenous dot product for 4-vector	2	1
86	0x56	dp3	3-wide dot product for 4-vector	2	1
87	0x57	dp2	2-wide dot product for 4-vector	2	1
88	0x58	*Reserved*			
89	0x59	line	Component-wise line equation computation (a multiply-add)	2	1
90	0x5A	pln	Component-wise floating point plane equation computation (a multiply-multiply-add)	2	1
91	0x5B	fma(mad)	Component-wise floating point mad computation (a multiple-add)	3	1

Opcode		Instruction	Description	#src	#dst
dec	hex				
92	0x5C	lrp	Component-wise floating point lrp computation (blend)	3	1
93	0x5D	***fmam(madm)***	Component-wise floating point fused multiply and add for macro operations.	3	1
94-95	0x5E-0x5F	*Reserved*			

Special Instructions

There are two special instructions, namely, *nop* (opcode = 0x7E) and *illegal* (opcode = 0x00).

- *Nop* instruction may be used for instruction padding in memory between two normal instructions to force alignment or to introduce instruction execution delay. Currently, there is no need for between-instruction padding.
- *Illegal* instruction may be used for instruction padding in memory outside the normal instruction sequence such as before or after the kernel program as well as between subroutines.
- *Nop* and *illegal* instructions do not have source operands or destination operand. Therefore, they do not implicitly update the accumulator register. They cannot be compressed.

Special Instructions

Opcode		Instruction	Description	#src	#dst
dec	hex				
0	0x00	illegal	Illegal instruction	0	0
96-125	0x60-0x7D	*Reserved*			
126	0x7E	nop	No-op	0	0
127	0x7F	*Reserved*			

Native Instruction BNF

The Backus-Naur Form (BNF) grammar identifies the assembly language syntax, which is native to the hardware. It does not include intelligent defaults, assembler pragmas, etc.

Instruction Groups

<Instruction>::=<UnaryInstruction>

<UnaryInstruction>::= <Predicate> <UnaryInst> <ExecSize> dst <SrcAccImm> <InstOptions>

<UnaryInst>::= <UnaryOp> <ConditionalModifier> <Saturate>

<UnaryOp>::= "*mov*" | "*frc*" | "*rndu*" | "*rndd*" | "*rnde*" | "*rndz*" | "*not*" | "*lzd*"

<BinaryInstruction>::= <Predicate> <BinaryInst> <ExecSize> dst <Src> <SrcImm> <InstOptions>

<BinaryInst>::= <BinaryOp> <ConditionalModifier> <Saturate>

<BinaryOp>::= "**mul**" | "**mac**" | "**mach**" | "**line**"" | "**pln**"

|"**sad2**" | "**sada2**" | "**dp4**" | "**dph**" | "**dp3**" | "**dp2**"" | "**lrp**"" | "**bfi1**"" | "**addc**"" | "**subb**"

<BinaryAccInstruction>::= <Predicate> <BinaryAccInst> <ExecSize> dst <SrcAcc> <SrcImm> <InstrOptions>

<BinaryAccInst>::= <BinaryAccOp> <ConditionalModifier> <Saturate>

<BinaryAccOp>::= "**avg**" | "**add**" | "**sel**"

|"**and**" | "**or**" | "**xor**"

|"**shr**" | "**shl**" | "**asr**"

|"**cmp**" | "**cmpn**"

<TriInstruction>::= <Predicate> <TriInst> <ExecSize> <PostDst> <CurrDst> <TriSrc> <MsgDesc> <InstOptions>

<TriInst>::= <TriOp> <ConditionalModifier> <Saturate>

<TriOp>::= "**send**"

<TriInstruction2> ::= <Predicate> <TriInst2> <ExecSize> dst <Src> <Src> <Src><InstOptions>

<TriInst2> ::= <TriOp> <ConditionalModifier><Saturate>

,<TriOp> ::= "bfe"| "bfi2"|"mad"

<BranchConvInstruction> ::= <Predicate> <BranchConvOp> <ExecSize>< RelativeLocation2>

<BranchConvOp> ::= "brc"

<BrancConvInstruction> ::= <Predicate> <BranchDivOp> <ExecSize>< RelativeLocation3>

<BranchDivOp> ::= "brd"

<CallInstruction> ::= <Predicate> <CallOp> <ExecSize>dst< RelativeLocation2>

<CallOp> ::= “call” |”CALLA”

<MathInstruction> ::= <Predicate> <MathInst> <ExecSize>< Dst>< Src>< Src><FC>

<MathInst> ::= <MathOp><Saturate>

<MathOp> ::= “math”

<FC> ::= “**INV**” | “**LOG**” | “**EXP**” | “**SQRT**” |”**RSQ**” | “**POW**” | “**SIN**” | “**COS**” | “**INT DIV**”

<JumpInstruction> ::=<JumpOp> <RelativeLocation2>

<JumpOp>::= “**jmpi**”

<BranchLoopInstruction>::= <Predicate> <BranchLoopOp> < RelativeLocation>

<BranchLoopOp>::=“**if**” | “**iff**” | “**while**”

<ElseInstruction>::= <ElseOp> < RelativeLocation>

<ElseOp>::=“**else**”

<BreakInstruction>::= <Predicate> <BreakOp> <LocationStackCtrl>

<BreakOp>::=“**break**” | “**cont**” | “**halt**”

<SyncInstruction>::= <Predicate> <SyncOp> <NotifyReg>

<SyncOp>::=“**wait**”

<SpecialInstruction>::=“**do**” | “**endif**” |”**nop**” | “**illegal**”

Destination Register

dst::=<DstOperand>

<DstOperandEx>

<DstOperand>::=<DstReg> <DstRegion> <WriteMask> <DstType>

<DstOperandEx>::=<AccReg> <DstRegion> <DstType>

|<FlagReg> <DstRegion> <DstType>

|<AddrReg> <DstRegion> <DstType>

|<MaskReg> <DstRegion> <DstType>

|<MaskStackReg>

|<ControlReg>

|<IPReg>

|<NullReg>

| <ChannelEnableReg>

|<ThreadControlReg>

|<PerformanceReg>

<DstReg>::=<DirectGenReg> | <IndirectGenReg>

|<DirectMsgReg> | <IndirectMsgReg>

<PostDst>::=<PostDstReg> <DstRegion> <WriteMask> <DstType>

|<NullReg>

<PostDstReg>::= <DirectGenReg> | <IndirectGenReg>

<CurrDst>::=<DirectAlignedMsgReg>

Source Register

Source with Accumulator Access and with Immediate

<SrcAccImm>::=<SrcAcc>

<Imm32> <SrcImmType>

<SrcAcc>::=<DirectSrcAccOperand>

<IndirectSrcOperand>

<DirectSrcAccOperand>::=<DirectSrcOperand>

|<SrcArcOperandEx>

|<AccReg> <SrcType>

<SrcArcOperandEx>::=<FlagReg> <Region> <SrcType>

|<AddrReg> <Region> <SrcType>

|<ControlReg>

|<StateReg>

|<NotifyReg>

|<IPReg>

|<NullReg>

| <ChannelEnableReg>

|<ThreadControlReg>

|<PerformanceReg>

<IndirectSrcOperand>::=<SrcModifier> <IndirectGenReg> <IndirectRegion> <Swizzle > <SrcType>

Source without Accumulator Access

<Src>::=<DirectSrcOperand>

<IndirectSrcOperand>

< DirectSrcOperand>::=<SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>

|<SrcArcOperandEx>

<TriSrc>::=<SrcModifier> <DirectGenReg> <Region> <Swizzle> <SrcType>

<NullReg>

<MsgDesc>::=<ImmDesc>

<Reg32>

<Reg32>::=<DirectGenReg> <Region> <SrcType>

Source without Accumulator Access or IP Access

<SrcImm>::=<DirectSrcOperand>

<Imm32> <SrcImmType>

Address Registers

<AddrParam>::=<AddrReg> <ImmAddrOffset>

<ImmAddrOffset>::= ""

| "**,**" <ImmAddrNum>

Register Files and Register Numbers

Note: The recommended instruction syntax uses subregister numbers within the GRF in units of actual data element size, corresponding to the data type used. For example for the F (Float) type, the assembler syntax uses subregister numbers 0 to 7, corresponding to subregister byte addresses of 0 to 28 in steps of 4, the element size.

<DirectGenReg>::=<GenRegFile> <GenRegNum> <GenSubRegNum>

<IndirectGenReg>::=<GenRegFile> "**[**" <AddrParam> "**]**"

<GenRegFile>::="**r**"

<GenRegNum>:: ="**0**"…"**127**"

<GenSubRegNum>:: = ""

| ".0"…".3" //incase of DF

| "**.0**"…"**.7**"

| "**.0**"…"**.15**"

| "**.0**"…"**.31**"

<DirectMsgReg>::=<DirectAlignedMsgReg> <MsgSubRegNum>

<DirectAlignedMsgReg>::=<MsgRegFile> <MsgRegNum>

<IndirectMsgReg>::=<MsgRegFile> "**[**" <AddrParam> "**]**"

<MsgRegFile>::="**m**"

<MsgRegNum>:: =“**0**”…“**15**”

<MsgSubRegNum>:: = <GenSubRegNum>

<AddrReg>::=<AddrRegFile> <AddrSubRegNum>

<AddrRegFile>::=“**a0**”

<AddrSubRegNum>:: = “”

| “**.0**” … “**.7**”

<AccReg>::=“**acc**” <AccRegNum><AccSubRegNum>

<AccRegNum>:: =“**0**” | “**1**”

<AccSubRegNum>:: = <GenSubRegNum>

<FlagReg> ::= “f” <FlagRegNum> <FlagSubRegNum>

<FlagRegNum> :: = “0” | “1”

<FlagReg>::=“**f0**” <FlagSubRegNum>

<FlagSubRegNum>:: =“”

| “**.0**”…“.**1**”

<NotifyReg>::=“**n**” <NotifyRegNum>

<NotifyRegNum>:: =“**0**”…“**2**”

<StateReg>::=“**sr0**” <StateSubRegNum>

<StateSubRegNum>:: =“.**0**”… “.**1**”

<ControlReg>::=“**cr0**” <ControlSubRegNum>

<ControlSubRegNum>:: =“.**0**” …“.**2**”

<IPReg>::=“**ip**”

<NullReg>::=“**null**”

<ThreadControlReg> ::= “tdr0”<ThreadCntrlSubRegNum>

<ThreadCntrlSubRegNum> ::= “.0”…”.7”

<PerformanceReg> ::= “tm0”

<ChannelEnableReg> ::= “ce0.0”

Relative Location and Stack Control

<RelativeLocation>::= <imm16>

<RelativeLocation2>::= <imm32> | <reg32>

<RelativeLocation3> ::= <imm16> | <reg32>

<LocationStackCtrl>::=<imm32>

Regions

<DstRegion>::=“**<**”<HorzStride> “**>**”

<IndirectRegion>::=<Region> | <RegionWH> | <RegionV>

<Region>::=“**<**”<VertStride> “**;**” <Width> “**,**” <HorzStride> “**>**”

<RegionWH>::=“**<**” <Width> “**,**” <HorzStride> “**>**”

<RegionV>::=“**<**”<VertStride> “**>**”

<VertStride>::= “**0**” | “**1**” | “**2**” | “**4**” | “**8**” | “**16**” | “**32**”

<Width>::=“**1**” | “**2**” | “**4**” | “**8**” | “**16**”

<HorzStride>::=“**0**” | “**1**” | “**2**” | “**4**”

Types

SrcType
<SrcType> ::= “**:df**” | “**:f**” | “**:ud**” | “**:d**” | “**:uw**” | “**:w**” | “**:ub**” | “**:b**” | “**:uq**” | “**:q**” | “**:hf**”

<SrcImmType> ::= <SrcType> | “**:v**” | “**:vf**” | “**:uv**”

<DstType> ::= <SrcType>

Write Mask

<WriteMask>::= “”

| “.” “**x**” | “.” “**y**” | “.” “**z**” | “.” “**w**”

| “.” “**xy**” | “.” “**xz**” | “.” “**xw**” | “.” “**yz**” | “.” “**yw**” | “.” “**zw**”

| “.” “**xyz**” | “.” “**xyw**” | “.” “**xzw**” | “.” “**yzw**”

| “.” “**xyzw**”

Swizzle Control

<Swizzle>::=“”

| “.” <ChanSel>

| “.” <ChanSel> <ChanSel> <ChanSel> <ChanSel>

<ChanSel>::= “**x**” | “**y**” | “**z**” | “**w**”

Immediate Values

<ImmAddrNum>::=“-**512**”... “**511**”

<Imm64> ::= “0.0”... “±1.0*2-1024...1023” | “0”...”264-1” | “-263”...”263-1”

<Imm32>::=“**0.0**”... “**±1.0*2**$^{-128...127}$” | “**0**”...”**2**^{32}**-1**” | “-**2**^{31}”...”**2**^{31}**-1**”

<Imm16>::="**0**"…"**2^16-1**" | "-**2^15**"…"**2^15-1**"

<ImmDesc>::="**0**"…"**2^32-1**"

Predication and Modifiers

Instruction Predication

<Predicate>::=""

|"**(**" <PredState> <FlagReg> <PredCntrl> "**)**"

<PredState>::=""

|"**+**"

|"**-**"

<PredCntrl>::=""

|"**.x**" | "**.y**" | "**.z**" | "**.w**"

|"**.any2h**" | "**.all2h**"

|"**.any4h**" | "**.all4h**"

|"**.any8h**" | "**.all8h**"

|"**.any16h**" | "**.all16h**"

|"**.anyv**" | "**.allv**"

| "**.any32h**"| "**.all32h**"

Source Modification

<SrcModifier**>**::=""

|"**-**"

|"**(abs)**"

|"**-**" "**(abs)**"

Instruction Modification

<ConditionalModifier>::=""

|<CondMod> "**.** " <FlagReg>

<CondMod>::="**.z**" | "**.e**"|"**.nz**" | "**.ne**"|"**.g**"|"**.ge**"|"**.l**"|"**.le**"|"**.o**" |"**.r**" |"**.u**"

<Saturate>::=""

|"**.sat**"

Execution Size

<ExecSize>::="**(**" <NumChannels> "**)**"

<NumChannels>::="**1**" | "**2**" | "**4**" | "**8**" | "**16**" | "**32**"

Instruction Options

<InstOptions> ::= ""

| "**{**" <InstOption> "**}**"

| "**{**" <InstOption> <InstOptionEx> "**}**"

<InstOptionEx> ::= ""

| "**,**" <InstOption> <InstOptionEx>

<InstOption> ::= <AccessMode>

| <AccWrCtrl>

| <ComprCtrl>

| <DependencyCtrl>

| <MaskCtrl>

| <SendCtrl>

| <ThreadCtrl>

<AccessMode> ::= "**Align1**" | "**Align16**"

<AccWrCtrl> ::= "**AccWrEn**"

<ComprCtrl> ::= "**SecHalf**" | "**Compr**"

<DependencyCtrl> ::= "**NoDDClr**"

<MaskCtrl> ::= "**NoMask**"

<SendCtrl> ::= "**EOT**"

<ThreadCtrl> ::= "**Switch**"

| "**Atomic**"

Note for Assembler: Compression control "**Compr**" has a direct map to the binary instruction word. It may be omitted if the Assembler can determine whether an instruction is compressable.

Instruction Set Summary Tables

The columns in the following tables specify instruction mnemonics, hex opcodes, full names, instruction groups, processor generation (where blank means available), the number of source operands, whether the instruction supports predication, any support for source modifiers, an indication of supported data types, whether the instruction supports saturation, and any support for conditional modifiers.

See the separate Accumulator Restrictions table for information about how instructions are allowed to use accumulators.

If the Project column is blank, an instruction is supported for CHV, BSW, all generations implemented or designed so far.

N and Y indicate No (no support for a feature) and Yes (full support for a feature) respectively.

A SrcMod (source modifier) value of Y indicates that a numeric source modifier is allowed, optionally specifying abolute value, negation, or a forced negative value. The value N indicates no source modifier support.

SrcMod Information
A SrcMod value of ** indicates a logical source modifier is allowed, optionally inverting all source bits (a NOT operation).

In the Src Types and Dst Type columns, Int means any integer type and * means such an extensive list of types that you must refer to the detailed instruction description.

Byte (B, UB) and QWord (Q, UQ) integer types cannot be mixed in the same instruction.

Instruction Set Summary Table A to B (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
add	40	Addition	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
addc	4E	Integer Addition with Carry	Parallel Arithmetic	2	Y	N	UD	UD	N	Y
and	05	Logic And	Move and Logic	2	Y	**	Int	Int	N	Equality only
asr	0C	Arithmetic Shift Right	Move and Logic	2	Y	Y	Int	Int	Y	Y
avg	42	Average	Parallel Arithmetic	2	Y	Y	B, UB W, UW D, UD	B, UB W, UW D, UD	Y	Y
bfe	18	Bit Field Extract	Move and Logic	3	Y	N	UD, D	UD, D	N	N
bfi1	19	Bit Field Insert 1	Move and Logic	2	Y	N	UD, D	UD, D	N	N
bfi2	1A	Bit Field Insert 2	Move and Logic	3	Y	N	UD, D	UD, D	N	N
bfrev	17	Bit Field Reverse	Move and Logic	1	Y	N	UD	UD	N	N
brc	23	Branch Converging	Flow Control	0 or 1	Y	N	D		N	N
brd	21	Branch Diverging	Flow Control	0 or 1	Y	N	D		N	N
break	28	Break	Flow Control	0	Y	N			N	N

Instruction Set Summary Table C to E (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
call	2C	Call	Flow Control	0	Y	N		D, UD	N	N
calla	2B	Call Absolute	Flow Control	0	Y	N		D, UD	N	N
cbit	4D	Count Bits Set	Move and Logic	1	Y	N	UB, UW, UD	UD	N	N
cmp	10	Compare	Move and Logic	2	Y	Y	*	*	N	Y
cmpn	11	Compare NaN	Move and Logic	2	Y	Y	*	*	N	Y
cont	29	Continue	Flow Control	0	Y	N			N	N
csel	12	Conditional Select	Move and Logic	3	N	Y	F	F	Y	Y
dp2	57	Dot Product 2	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dp3	56	Dot Product 3	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dp4	54	Dot Product 4	Vector Arithmetic	2	Y	Y	F	F	Y	Y
dph	55	Dot Product Homogeneous	Vector Arithmetic	2	Y	Y	F	F	Y	Y
else	24	Else	Flow Control	0	N	N			N	N
endif	25	End If	Flow Control	0	N	N			N	N

Instruction Set Summary Table F to L (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
fbh	4B	Find First Bit from MSB Side	Move and Logic	1	Y	N	D, UD	UD	N	N
fbl	4C	Find First Bit from LSB Side	Move and Logic	1	Y	N	UD	UD	N	N
frc	43	Fraction	Parallel Arithmetic	1	Y	Y	F	F	N	Y
goto	2E	Goto	Flow Control	0	Y	N			N	N
halt	2A	Halt	Flow Control	0	Y	N			N	N
if	22	If	Flow Control	0	Y	N			N	N
illegal	00	Illegal	Special	0	N	N			N	N
jmpi	20	Jump Indexed	Flow Control	1	Y	N	D		N	N
join	2F	Join	Flow Control	0	Y	N			N	N
line	59	Line	Vector Arithmetic	2	Y	Y	F	F	Y	Y
lrp	5C	Linear Interpolation	Vector Arithmetic	3	Y	Y	F	F	N	Y
lzd	4A	Leading Zero Detection	Move and Logic	1	Y	Y	D, UD	UD	Y	Y

Instruction Set Summary Table M to P (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
mac	48	Multiply Accumulate	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
mach	49	Multiply Accumulate High	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
mad	5B	Multiply Add	Vector Arithmetic	3	Y	Y	*	*	Y	Y
madm	5D	Multiply Add for Macro	Vector Arithmetic	3	Y	Y	*	*	Y	Y
math	38	Extended Math Function	Miscellaneous	2	Y	N	*	*	Y	N
mov	01	Move	Move and Logic	1	Y	Y	*	*	Y	Y
movi	03	Move Indexed	Move and Logic	1	Y	Y	*	*	Y	N
mul	41	Multiply	Parallel Arithmetic	2	Y	Y	*	*	Y	Y
nop	7E	No Operation	Special	0	N	N			N	N
not	04	Logic Not	Move and Logic	1	Y	**	Int	Int	N	Equality only
or	06	Logic Or	Move and Logic	2	Y	**	Int	Int	N	Equality only

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
pln	5A	Plane	Vector Arithmetic	2	Y	Y	F	F	Y	Y

Instruction Set Summary Table R to X (Listed by Instruction Mnemonic)

Mnem.	Hex Opcode	Name	Group	Srcs	Pred?	SrcMod	Src Types	Dst Type	Sat?	CondMod?
ret	2D	Return	Flow Control	1	Y	N	D, UD		N	N
rndd	45	Round Down	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rnde	46	Round to Nearest or Even	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rndu	44	Round Up	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
rndz	47	Round to Zero	Parallel Arithmetic	1	Y	Y	F	F	Y	Y
sad2	50	Sum of Absolute Difference 2	Vector Arithmetic	2	Y	Y	B, UB	W, UW	Y	Y
sada2	51	Sum of Absolute Difference Accumulate 2	Vector Arithmetic	2	Y	Y	B, UB	W, UW	Y	Y
sel	02	Select	Move and Logic	2	Y	Y	*	*	Y	Y
send	31	Send Message	Miscellaneous	1	Y	N	*	*	N	N
sendc	32	Conditional Send Message	Miscellaneous	1	Y	N	*	*	N	N
shl	09	Shift Left	Move and Logic	2	Y	Y	Int	Int	Y	Y
shr	08	Shift Right	Move and Logic	2	Y	Y	Int	Int	Y	Y
smov	0A	Scattered Move	Move and Logic	1	Y	N	*	*	N	N
subb	4F	Integer Subtraction with Borrow	Parallel Arithmetic	2	Y	N	UD	UD	N	Y
wait	30	Wait	Miscellaneous	1	N	N	UD	UD	N	N
while	27	While	Flow Control	0	Y	N			N	N
xor	07	Logic Xor	Move and Logic	2	Y	**	Int	Int	N	Equality only

>

Accumulator Restrictions

This section describes restrictions on accumulator access: general restrictions, restrictions for specific instructions, and how those specific restrictions vary for processor generations. See Accumulator Registers for a description of the accumulator registers.

Accumulator registers can be accessed as explicit source or destination operands, as an implicit source value when specified for a particular instruction (*sada2* for example), and as an implicit destination when the AccWrEn instruction option is used.

These general rules apply to accumulator access:

1. Flow control, *send*, *sendc*, and *wait* instructions cannot use accumulators.
2. Instructions with three source operands cannot use explicit accumulator operands. AccWrEn may be allowed for implicitly updating the accumulator.
3. Instructions that use the accumulator as an implicit source value cannot specify an explicit accumulator source operand.
4. Instructions that specify an implicit accumulator destination (with AccWrEn) cannot specify an explicit accumulator destination operand.
5. An instruction with both an explicit accumulator source operand and an explicit accumulator destination operand must specify the same accumulator register as the source and the destination.

These descriptions are frequently used in this table:

- No restrictions.
- No accumulator access, implicit or explicit.
- Source operands cannot be accumulators.
- Source modifier is not allowed if source is an accumulator.
- Accumulator is an implicit source and thus cannot be an explicit source operand.
- Accumulator cannot be destination, implicit or explicit.
- AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.

These minor cases occur occasionally in the table:

- Integer source operands cannot be accumulators.
- No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.
- An accumulator can be a source or destination operand but not both.

A few instructions use more than one of the listed restrictions.

Accumulator Restrictions

Instructions	Accumulator Restrictions
add	No restrictions.
addc	AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
and	Source modifier is not allowed if source is an accumulator.
asr *avg*	No restrictions.
bfe *bfi1* *bfi2* *bfrev* *cbit*	No accumulator access, implicit or explicit.
cmp	No restrictions.
cmpn	No restrictions.
csel	No restrictions.
dp2 *dp3* *dp4* *dph*	Source operands cannot be accumulators.
fbh *fbl*	No accumulator access, implicit or explicit.
frc	No restrictions.
line	Source operands cannot be accumulators.
lrp	No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.
lzd	No restrictions.
mac	Accumulator is an implicit source and thus cannot be an explicit source operand.
mach	Accumulator is an implicit source and thus cannot be an explicit source operand. AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
madm	No explicit accumulator access because this is a three-source instruction. AccWrEn is allowed for implicitly updating the accumulator.

Instructions	Accumulator Restrictions
math	No accumulator access, implicit or explicit.
mov	An accumulator can be a source or destination operand but not both.
movi	Source operands cannot be accumulators.
mul	Integer source operands cannot be accumulators.
not *or*	Source modifier is not allowed if source is an accumulator.
pln	Source operands cannot be accumulators.
rndd *rnde* *rndu* *rndz*	No restrictions.
sad2 *sada2*	Source operands cannot be accumulators.
sel	No restrictions.
shl	Accumulator cannot be destination, implicit or explicit.
shr	No restrictions.
smov	No restrictions.
subb	AccWrEn is required. The accumulator is an implicit destination and thus cannot be an explicit destination operand.
xor	Source modifier is not allowed if source is an accumulator.

Instruction Set Reference

This chapter describes the functions of 3D Media GPGPU Execution Units, listed in alphabetical order according to assembly language mnemonic.

EUISA Instructions List

Conventions

This section describes conventions used in instruction reference pages.

For each instruction that has source or destination types, a table lists the allowed type combinations and may also indicate the processor generations that support certain combinations. A notation like *W indicates that UW and W are both allowed. Multiple types listed together mean that any combination (Cartesian product) of the listed types is allowed.

If a source operand is floating-point, all source operands must have the same floating-point data type.

The Q and UQ types cannot be mixed with the B or UB types, neither as different source types nor as source type and destination type.

Accumulator restrictions are described in the Accumulator Restrictions section and also appear in instruction descriptions.

Pseudo Code Format

Instructions are explained in the following pseudo-code format that resembles the GEN assembly instruction format.

```
[(pred)] opcode (exec_size) dst src0 [src1]
```

Square brackets "[ ]" indicate that a field is optional. Saturation modifiers and instruction options are omitted for simplicity.

General Macros and Definitions

INST_MIN_SIZE is defined as a constant of 8 bytes.

```
#define INST_MIN_SIZE  8  // Instruction minimum size in bytes (for the compact instruction format)
```

The floor function converts a floating point value to an integral floating point value. For a given floating point value, from its closest two integral float values, floor returns the one that is closer to negative infinity. For example, floor(1.3f) = 1.0f and floor(-1.3f) = -2.0f.

```
float floor(float g)
{
    return maximum(any integral float f: f <= g)
}
```

The Condition function takes the conditional signals {SN, ZR, OF, IN, NC} of result, generates a Boolean value according to a conditional evaluation controlled by the conditional modifier cmod, and returns the Boolean.

```
Bool Condition(result, cmod)
```

The ConditionNaN function takes the conditional signals {SN, ZR, OF, IN, NC, NS} of result, generates a Boolean value according to a conditional evaluation controlled by the conditional modifier cmod, and returns the Boolean. The only difference between Condition and ConditionNaN is that ConditionNaN uses the NS (NaN of the second source) signal.

```
Bool ConditionNaN(result, cmod)
```

The Jump function jumps the instruction sequence from the current instruction location by InstCount 8-byte units, where each 16-byte native instruction is two units and each 8-byte compact instruction is one unit. If InstCount is positive and greater than zero, is an unconditional jump forward. If InstCount is negative, is an unconditional jump backward. If InstCount is zero, IP stays on the current instruction in an infinite loop.

```
void Jump(int InstCount)
{
    IP = IP + (InstCount * INST_MIN_SIZE)
}
```

Evaluate Write Enable

The WrEn should be evaluated as below.

Note: MaskCtrl = NoMask (1) skips the check for PcIP[n] == ExIP before enabling a channel.

```
if ( MaskCtrl == 1 ) {
  for ( n = 0; n < exec_size; n++ ) {
    WrEn[n] = 1;
  }
}
else {
  for ( n = 0; n < exec_size; n++ ) {
    if ( PcIP[n] == ExIP ) {
      WrEn[n] = 1;
    }
    else {
      WrEn[n] = 0;
    }
  }
}

if ( PredCtrl != 0000b ) {
  for ( n = 0; n < exec_size; n++ ) {
    WrEn[n] = WrEn[n] & PMask[n];
  }
}

for ( n = exec_size; n < 32; n++ ) {
  WrEn[n] = 0;
}
```

EUISA Instructions

Symbol	Name	Source
add	**Addition**	EUISA
addc	**Addition with Carry**	EUISA
asr	**Arithmetic Shift Right**	EUISA
avg	**Average**	EUISA
bfe	**Bit Field Extract**	EUISA
bfi1	**Bit Field Insert 1**	EUISA
bfi2	**Bit Field Insert 2**	EUISA
bfrev	**Bit Field Reverse**	EUISA
brc	**Branch Converging**	EUISA
brd	**Branch Diverging**	EUISA
break	**Break**	EUISA
call	**Call**	EUISA
calla	**Call Absolute**	EUISA
cmp	**Compare**	EUISA
cmpn	**Compare NaN**	EUISA
csel	**Conditional Select**	EUISA
sendc	**Conditional Send Message**	EUISA
cont	**Continue**	EUISA
cbit	**Count Bits Set**	EUISA
dp2	**Dot Product 2**	EUISA
dp3	**Dot Product 3**	EUISA
dp4	**Dot Product 4**	EUISA
dph	**Dot Product Homogeneous**	EUISA
else	**Else**	EUISA
endif	**End If**	EUISA
math	**Extended Math Function** • INV - Inverse • LOG – Logarithm • EXP - Exponent • SQRT - Square Root • RSQ - Reciprocal Square Root • POW - Power Function • SIN - SINE • COS - COSINE	EUISA

	• INT DIV - Integer Divide • INVM/RSQRTM [CHV, BSW]	
fbl	**Find First Bit from LSB Side**	EUISA
fbh	**Find First Bit from MSB Side**	EUISA
frc	**Fraction**	EUISA
goto	**Goto**	EUISA
halt	**Halt**	EUISA
if	**If**	EUISA
illebal	**Illegal**	EUISA
subb	**Integer Subtraction with Borrow**	EUISA
join	**Join**	EUISA
jmpi	**Jump Indexed**	EUISA
lzd	**Leading Zero Detection**	EUISA
line	**Line**	EUISA
lrp	**Linear Interpolation**	EUISA
and	**Logic And**	EUISA
not	**Logic Not**	EUISA
or	**Logic Or**	EUISA
xor	**Logic Xor**	EUISA
mov	**Move**	EUISA
movi	**Move Indexed**	EUISA
mul	**Multiply**	EUISA
mac	**Multiply Accumulate**	EUISA
mach	**Multiply Accumulate High**	EUISA
mad	**Multiply Add**	EUISA
madm	**Multiply Add for Macro**	EUISA
nop	**No Operation**	EUISA
pln	**Plane**	EUISA
ret	**Return**	EUISA
rndd rnde rndu rndz	**Round Instructions** ▪ **Round Down** ▪ **Round to Nearest or Even** ▪ **Round Up** ▪ **Round to Zero**	EUISA
smov	**Scattered Move**	EUISA
sel	**Select**	EUISA
send	**Send Message**	EUISA

shl	**Shift Left**	EUISA
shr	**Shift Right**	EUISA
sad	**Sum of Absolute Difference 2**	EUISA
sada2	**Sum of Absolute Difference Accumulate 2**	EUISA
wait	**Wait Notification**	EUISA
while	**While**	EUISA

Round Instructions

rndd - Round Down

rndu - Round Up

rnde - Round to Nearest or Even

rndz - Round to Zero

rndd – Round Down

Description:

The *rndd* instruction takes component-wise floating point downward rounding (to the integral float number closer to negative infinity) of src0 and storing the rounded integral float results in dst. This is commonly referred to as the floor() function.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round Down in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	-finite	^	–0	+0	+0	**	+inf	NaN
Notes:									
^	Note: {-1, -0} depending on the Single Precision Denorm Mode.								
**	Result may be {+finite, +0}.								

Floating-Point Round Down in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	-finite	–0	–0	+0	+0	**	+fmax	
Notes:									
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rnde – Round to Nearest or Even

Description:

The *rnde* instruction takes component-wise floating point round-to-even operation of src0 with results in two pieces – a downward rounded integral float results stored in dst and the round-to-even increments stored in the rounding increment bits. The round-to-even increment must be added to the results in dst to create the final round-to-even values to emulate the round-to-even operation, commonly known as the round() function. The final results are the one of the two integral float values that is nearer to the input values. If the neither possibility is nearer, the even alternative is chosen.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round to Nearest or Even in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	+0	**	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								

Floating-Point Round to Nearest or Even in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	**	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rndu – Round Up

Description:

The *rndu* instruction takes component-wise floating point upward rounding (to the integral float number closer to positive infinity) of src0, commonly known as the ceiling() function.

Each result follows the rules in the following tables based on the floating-point mode.

Floating-Point Round Up in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	^	+finite	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
^	Note {+1, +0} depending on the Single Precision Denorm Mode.								

Floating-Point Round Up in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	+finite	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

rndz – Round to Zero

Description:

The *rndz* instruction takes component-wise floating point round-to-zero operation of src0 with results in two pieces – a downward rounded integral float results stored in dst and the round-to-zero increments stored in the rounding increment bits. The round-to-zero increment must be added to the results in dst to create the final round-to-zero values to emulate the round-to-zero operation, commonly known as the truncate() function. The final results are the one of the two closest integral float values to the input values that is nearer to zero.

Floating-Point Round to Zero in IEEE mode

src0	–inf	–finite	–denorm	–0	+0	+denorm	+finite	+inf	NaN
dst	–inf	*	–0	–0	+0	+0	**	+inf	NaN
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								

Floating-Point Round to Zero in ALT mode

src0	–fmax	–finite	–denorm	–0	+0	+denorm	+finite	+fmax	***
dst	–fmax	*	–0	–0	+0	+0	**	+fmax	
Notes:									
*	Result may be {–finite, –0}.								
**	Result may be {+finite, +0}.								
***	Result is undefined if src0 is {–inf, +inf, NaN}.								

Send Message

send - Send Message

The following information describes the send message.

Send Message [CHV, BSW]

Opcode	Instruction	Description
49 (0x31)	send <dest> <src> <ex_desc> <desc>	Send a message stored in GRF starting at <src> to a shared function identified by <ex_desc> along with control from <desc> with a GRF writeback location at <dest>.

Pred	Sat	Cond Mod	Src Mod	Src Types	Dst Type
Y					[FLT] [INT]

Format:

```
[(pred)] send (exec_size) <dest> <src> <ex_desc> <desc>
```

Syntax:

Syntax
[(pred)] send (exec_size) reg greg imm6 reg32a [(pred)] send (exec_size) reg greg imm6 imm32

Pseudocode:

```
Evaluate(WrEn);
<MsgChEnable> = WrEn;
<SourceReg> = <src>.RegNum;
MessageEnqueue(<MsgChEnable>, <ResponseReg>, <SourceReg>, <ex_dest>, <desc>);
```

Description:

The *send* instruction performs data communication between a thread and external function units, including shared functions (Sampler, Data Port Read, Data Port Write, URB, and Message Gateway) and some fixed functions (e.g. Thread Spawner, who also have an unique Shared Function ID). The *send* instruction adds an entry to the EU's message request queue. The request message is stored in a block of contiguous GRF registers. The response message, if present, will be returned to a block of contiguous GRF registers. The return GRF writes may be in any order depending on the external function units. <src> is the lead GRF register for request. <dest> is the lead GRF register for response. The message descriptor field <desc> contains the Message Length (the number of consecutive GRF registers) and the Response Length (the number of consecutive GRF registers). It also contains the header present bit, and the function control signals. The extend mesage descriptor field <ex_desc> contains the target function ID. WrEn is forwarded to the target function in the message sideband.

The *send* instruction is the only way to terminate a thread. When the EOT (End of Thread) bit of <ex_desc> is set, it indicates the end of thread to the EU, the Thread Dispatcher and, in most cases, the parent fixed function.

Message descriptor field <desc> can be a 32-bit immediate, *imm32*, or a 32-bit scalar register, <reg32a>. GEN restricts that the 32-bit scalar register <reg32a> must be the leading dword of the address register. It should be in the form of a0.0<0;1,0>:ud. When <desc> is a register operand, only the lower 29 bits of <reg32a> are used.

Syntax
<ex_desc> is a 6-bit immediate, imm6. The lower 4bits of the <ex_desc> specifies the SFID for the message. The MSb of the message descriptor, the EOT field, always comes from bit 127 of the instruction word, which is the MSb of imm6. A thread must terminate with a *send* instruction with EOT turned on.

<src> is a 256-bit aligned GRF register. It serves as the leading GRF register of the request.

<dest> serves for two purposes: to provide the leading GRF register location for the response message if present, and to provide parameters to form the channel enable sideband signals.

<dest> signals whether there is a response to the message request. It can be either a null register, a direct-addressed GRF register or a register-indirect GRF register. Otherwise, hardware behavior is undefined.

If <dest> is null, there is no response to the request. Meanwhile, the Response Length field in <desc> must be 0. Certain types of message requests, such as memory write (store) through the Data Port, do not want response data from the function unit. If so, the posted destination operand can be null.

If <dest> is a GRF register, the register number is forwarded to the shared function. In this case, the target function unit must send one or more response message phases back to the requesting thread. The number of response message phases must match the Response Length field in <desc>, which of course cannot be zero. For some cases, it could be an empty return message. An empty return message is defined as a single phase message with all channel enables turned off.

The subregister number, horizontal stride, destination mask and type fields of <dest> are always valid and are used in part to generate on the WrEn. This is true even if <dest> is a null register (this is an exception for null as for most cases these fields are ignored by hardware).

The 16-bit channel enables of the message sideband are formed based on the WrEn. Interpretation of the channel enable sideband signals is subject to the target external function. In general for a 'send' instruction with return messages, they are used as the destination dword write mask for the GRF registers starting at <dest>. For a message that has multiple return phases, the same set of channel enable signals applies to all the return phases.

Thread managed memory coherency: A special usage of using non-null <dest> is to support write-commit signaling for memory write service by the Data Port Write unit. If <post_dest> is not null for a memory write request, the Data Port along with the Data Cache or Render Cache will wait until all the posted writes for the request have reached the coherent domain before sending back to the requesting thread an empty message to <dest> register. A memory write reaching the coherent domain, also referred to as reaching the global observable state, means that subsequent read to the same memory location, no matter which thread issues the read, must return the data of the write.

NoDDClr must not be used for send instruction.

[CHV, BSW] r127 must not be used for return address when there is a src and dest overlap in send instruction.

Message Descriptor Definition [CHV, BSW]

Bit	Description
31	Reserved : MBZ
30	**Data Format.** This field specifies the width of data read from sampler or written to render target. Format = U1 0 – Single Precision (32b) 1 – Half Precision (16b)
30	Reserved : MBZ
29	Reserved : MBZ
28:25	**Message Length.** This field specifies the number of 256-bit GRF registers starting from <src> to be sent out on the request message payload. Valid value ranges from 1 to 15. A value of 0 is considered erroneous. Format = U4 Range = [1,15]

24:20	**Response Length.** This field indicates the number of 256-bit registers expected in the message response. The valid value ranges from 0 to 16. A value 0 indicates that the request message does not expect any response. The largest response supported is 16 GRF registers. Format = U5 Range = [0,16]
19	**Header Present.** If set, indicates that the message includes a header. Depending on the target shared function, this field may be restricted to either enabled or disabled. Refer to the specific shared function section for details. Format = Enable
18:0	**Function Control** This field is intended to control the target function unit. Refer to the section on the specific target function unit for details on the contents of this field.

Extended Message Descriptor Definition

Bit	Description
31:16	Reserved
5	**End Of Thread** This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
4	reserved
3:0	**Target Function ID** This field indicates the function unit for which the message is intended. ***Refer to "GPU Overview" document for the mapping of Shared Function IDs***

Restrictions:

Software must obey the following rules in signaling the end of thread using the *send* instruction:

- The posted destination operand must be null.
- No acknowledgement is allowed for the *send* instruction that signifies the end of thread. This is to avoid deadlock as the EU is expecting to free up the terminated thread's resource.
- A thread must terminate with a *send* instruction with message to a shared function on the output message bus; therefore, it cannot terminate with a *send* instruction with message to the following shared functions: Sampler unit, NULL function. For example, a thread may terminate with a URB write message or a render cache write message.
- A root thread originated from the media (generic) pipeline must terminate with a *send* instruction with message to the Thread Spawner unit. A child thread should also terminate with a send to TS. Please refer to the Media Chapter for more detailed description.

- The *send* instruction can not update accumulator registers.
- Saturate is not supported for *send* instruction.
- ThreadCtrl are not supported for *send* instruction.
- The *send* with EOT should use register space R112-R127 for <src>. This is to enable loading of a new thread into the same slot while the message with EOT for current thread is pending dispatch.
- SEND should not use 64-bit datatype for src or dest register.
- When context save and restore of a thread is required, the registers r0-r4 must not be used as *dest or src* registers. This is required to provide a grf register space to save sr1.0 by the system interrupt routine in the event of a mid thread pre-emption.

Extended Message Descriptor Definition

Bits	Description
5	**End Of Thread**This field, if set, indicates that this is the final message of the thread and the thread's resources can be reclaimed.
4	reserved
3:0	**Target Function ID** This field indicates the function unit for which the message is intended. Refer to volume GPU Overview for GPE Function IDs.

EUISA Structures

Name	Source
AddrSubRegNum	Eulsa
DstRegNum	Eulsa
DstSubRegNum	Eulsa
EU_INSTRUCTION_BASIC_ONE_SRC	Eulsa
EU_INSTRUCTION_BASIC_THREE_SRC	Eulsa
EU_INSTRUCTION_BASIC_TWO_SRC	Eulsa
EU_INSTRUCTION_BRANCH_CONDITIONAL	Eulsa
EU_INSTRUCTION_BRANCH_ONE_SRC	Eulsa
EU_INSTRUCTION_BRANCH_TWO_SRC	Eulsa
EU_INSTRUCTION_COMPACT_THREE_SRC	Eulsa
EU_INSTRUCTION_COMPACT_TWO_SRC	Eulsa
EU_INSTRUCTION_CONTROLS	Eulsa
EU_INSTRUCTION_CONTROLS_A	Eulsa

Name	Source
EU_INSTRUCTION_CONTROLS_B	Eulsa
EU_INSTRUCTION_HEADER	Eulsa
EU_INSTRUCTION_ILLEGAL	Eulsa
EU_INSTRUCTION_MATH	Eulsa
EU_INSTRUCTION_NOP	Eulsa
EU_INSTRUCTION_OPERAND_CONTROLS	Eulsa
EU_INSTRUCTION_OPERAND_DST_ALIGN1	Eulsa
EU_INSTRUCTION_OPERAND_DST_ALIGN16	Eulsa
EU_INSTRUCTION_OPERAND_SEND_MSG	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN1	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_ALIGN16	Eulsa
EU_INSTRUCTION_OPERAND_SRC_REG_THREE_SRC	Eulsa
EU_INSTRUCTION_SEND	Eulsa
EU_INSTRUCTION_SOURCES_IMM32	Eulsa
EU_INSTRUCTION_SOURCES_REG	Eulsa
EU_INSTRUCTION_SOURCES_REG_IMM	Eulsa
EU_INSTRUCTION_SOURCES_REG_REG	Eulsa
ExtMsgDescpt	Eulsa
FunctionControl	Eulsa
MsgDescpt31	Eulsa
SrcRegNum	Eulsa
SrcSubRegNum	Eulsa

EUISA Enumerations

Name	Source
AddrMode	Eulsa
ChanEn	Eulsa
ChanSel	Eulsa
CondModifier	Eulsa
DepCtrl	Eulsa
DstType	Eulsa
EU_OPCODE	Eulsa
ExecSize	Eulsa
FC	Eulsa
HorzStride	Eulsa
PredCtrl	Eulsa

Name	Source
QtrCtrl	Eulsa
RegFile	Eulsa
RepCtrl	Eulsa
SFID	Eulsa
SrcImmType	Eulsa
SrcIndex	Eulsa
SrcMod	Eulsa
SrcType	Eulsa
ThreadCtrl	Eulsa
VertStride	Eulsa
Width	Eulsa

EU Programming Guide

This topic is currently under development.

Assembler Pragmas

This topic is currently under development.

Declarations

A register or a register region can be declared as a symbol using the following form

.declare <symbol> **Base**=RegFile RegBase {.SubRegBase} **ElementSize**=ElementSize {**SrcRegion**=DefaultSrcRegion} {**DstRegion**=DefaultDstRegion} {**Type**=DefaultType}

The register file, the base of the register origin and the element size (in unit of bytes) are the mandatory parameters for a declared register region. Optionally, the base of the sub-register address, the default source region, the default destination region and the default type can be provided in the declaration for the symbol.

For immediate register addressing mode, the declared symbol can be used in the following Cartesian form

<symbol>(RegOff, SubRegOff)<=RegNum = *RegBase*+ **RegOff**; SubRegNum = *SubRegBase*+ **SubRegOff**

or in the following simplified row-aligned form

<symbol>(RegOff)<=RegNum = *RegBase*+ **RegOff;** SubRegNum = *SubRegBase*

For register-indirect-register-addressing mode, the declared symbol can be used to provide immediate address term in the following Cartesian form

<symbol>[IdxReg, RegOff, SubRegOff]<= RegNum (byte-aligned) = [**IdxReg]**+(*RegBase*+ **RegOff**)*32 + (*SubRegBase* + **SubRegOff**)**ElementSize*

or in the following simplified row-aligned form

<symbol>[IdxReg, RegOff]<= RegNum (byte-aligned) = [**IdxReg]**+(*RegBase*+ **RegOff**)*32

or in the form without the immediate address term

<symbol>[IdxReg]<= RegNum (byte-aligned) = [**IdxReg]**+ *RegBase*

Defaults and Defines

The default execution size is set according to the destination register type as the following

Destination Register Type	Default Execution Size
UB | B	(16)
UW | W	(16)
F | UD | D	(8)

The default execution size can be overwritten globally for all instructions using

.**default_execution_size***(Execution_Size)*

or be set according the **destination** register type using

.**default_execution_size**_*Type(Execution_Size)*

The default register type can be set for all register files using

.**default_register_type***Type*

or be set per register file using

.**default_register_type**_*RegFileType*

The default **source** register region for all symbols can be set using

.**default_source_register_region**<*VirtStride; Width, HorzStride*>

or be set per register type using

.**default_source_register_region**_*type*<*VirtStride; Width, HorzStride*>

The default **destination** register region for all symbols can be set using

.**default_destination_register_region**< *HorzStride*>

or be set per register type using

.**default_destination_register_region**_*type*< *HorzStride*>

Finally, the precompiler supports the string replacement statement of .define in the following form

.**define**<symbol>Expression

Programming Note	
Context:	Defaults and Defines
• **.declare** does not support nesting. In other words, each symbol in .declare must be self defined. This would allow the pre-processor to expand all symbols in one pass. • **.define** does support nesting. Only string substitution is supported (currently). • White space within square, angle and round brackets are allowed for easy source code alignment.	

Example Pragma Usages

Example: Declaration for 8x4=32-Byte Regions:

The following symbol Block can be used to address any 8x4 byte region within the Cartisian system of a 16x8 byte GRF register area starting from r0.

Declaration

```
// 32x4 Byte Array.declare BlockBase=r0 ElementSize=1 Region=<32;8,1>Type=b
```

Fully-Expressed Instr

```
mov(32)?:br0.16<32;8,1>:b// r0 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r1
xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r2 xxxxxxxxxxxxxxxxoooooooooxxxxxxxx// r3
xxxxxxxxxxxxxxxxoooooooooxxxxxxxx
```

Short-handed Instr

Mov?:bBlock(0,16)// (0,16): RegNum=0, SubRegNum=16

Example: Declaration for 8x1 Float Regions:

The following symbol *Trans* can be used to address any 8x1 float region within the Cartisian system of a 8x4 float GRF register area starting from r5.

Declaration

```
// 8x4 float Array starting at r5.declare Trans Base=r5 ElementSize=4 Region=<0;8,1> Type=f
```

Fully-Expressed Instr

mov(8)?:fr6.0<0;8,1>:f// 2nd 16x1 Row of Trans. Matrix // r5 FFFFFFFF// r6 OOOOOOOO// r7 FFFFFFFF// r8 FFFFFFFF

Short-handed Instr

mov?:fTrans(1) // RegNum = 5+1 = 6

Example: Declaration for 8x1 Float Regions with 1x1 Indirect Addressing:

Trans region defined (same as in the previous example) is used in conjunction with the address register.

Declaration

```
//8x4 float data array and 16x1 word address array.declare TransBase=r5 ElementSize=4
Region=<0;8,1> Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,224]<0;8,1>:f

Short-handed Instr

mov?:fTrans[a0.0,2] // [a0.0 + 5*32 + 2*32]

Example: Declaration with VxH Indirect Addressing:

The VxH register-indirect-register-addressing for Trans can be provided in the following short-hand form

Declaration

```
//8x4 float data array and word indices.declare TransBase=r5 ElementSize=4 Region=<0;8,1>
Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,224]<1,0>:f

Short-handed Instr

mov?:fTrans[a0.0,2]<1,0> // [a0.0+224] [a0.1+224] ... [a0.7+224]

Example: Declaration with Vx1 Indirect Addressing:

As width (4) is smaller than the execution region size (8), multiple indexed registers are used.

Declaration

```
//8x4 float data array and word address array.declare TransBase=r5 ElementSize=4
Region=<0;8,1> Type=f
```

Fully-Expressed Instr

mov(8)?:fr[a0.0,244]<4,1>:f

Short-handed Instr

```
mov?:fTrans[a0.0,2]<4,1>// [a0.0+224] [a0.1+224]
```

Assembly Programming Guideline

The following program skeleton illustrates the basic structure of a typical assembly program.

```
                          // single line comment

 /*          block comment
 */

 <preproc_directive>             // macros, include, etc.  Are global – handled by the pre-
processor
 <preproc_directive>             // applies to all code that follows in sequence

 // ------------ some kernel
 .kernel <kernel_name_string>    // [REQUIRED]

 // ------- Register requirements --------
 .reg_count_total    <uint>      // [REQUIRED] a more direct way to specify the parameters
required
 .reg_count_payload  <uint>      // [REQUIRED] rather than indirectly adding the
                          // the payload and temps together to get the total (as is the case
now)
                          // Note: no more "reg-count-temp"

 // -------------- Defaults ---------------
 <default…>                // these should be specified per-kernel and have only kernel-scope
 <default…>                // Same defaults as those already defined in the ISA doc, but just
 <default…>                // moved within the kernel to make each kernel completely self-
sufficient
                          // and not impacted defaults of earlier kernels

 // --------- Memory Requirements ---------
 // [optional] memory block info

 <MBDa>                    // memory block descriptor a
 <MBDb>                    // memory block descriptor b
 <MBDc>                    // memory block descriptor c
 <MBDd>                    // memory block descriptor d

 // ---------------- Code  ----------------
 .code                           // [REQUIRED]
   <instruction>
   <instruction>
   <instruction>
 <LabelLine>                     // labels are code-block scope
   <instruction>
   <instruction>
 .end_code                 // [REQUIRED]
```

```
.end_kernel                         // [REQUIRED]

// --------- next kernel -------------

// --------- next kernel -------------

// ...
```

Usage Examples

This topic is currently under development.

Vector Immediate

The immediate form of vector allows a constant vector to be in-lined in the instruction stream. An immediate vector is denoted by type v as *imm32:v*, where the 32-bit immediate field is partitioned into 8 4-bit subfields. Each 4-bit subfield contains a signed integer value. Therefore each 4-bit subfield has a range of [-8, +7]. This is depicted in the following figure.

31 28	27 24	23 20	19 16	15 12	11 8	7 4	3 0
V7	V6	V5	V4	V3	V2	V1	V0

Supporting DirectX 10 Pixel Shader Indexing

When a DirectX 10 Pixel Shader program is converted to run on GEN in channel-serial mode at 16 pixels in parallel, the per-pixel index must be translated into 16 indices with per channel offset. The creation of the per-channel offset can be achieved using the vector immediate.

Consider a generic DirectX 10 Pixel Shader instruction in the form of

opr4r[ind]r2

and assume that r0-r1 contain the 16 indices packed every other words, and r2-r3 contains source 1 and r4-r5 contain the destination. This instruction can be converted into the following GEN instructions. The corresponding operations are illustrated in *Supporting DirectX 10 Pixel Shader Indexing*.

mov (16) r11.0<1>:w 0x01234567:v// assigning a ramp vector, repeated once

mul (16)acc0:wr11.0<0;16,1>:w4:w// expand ramp range to 4 bytes per step

mac (16)r10.0<1>:wr0.0<16;8,2>:w32:w// r10 = index*32 + 0|4|...|28|0|4...|28

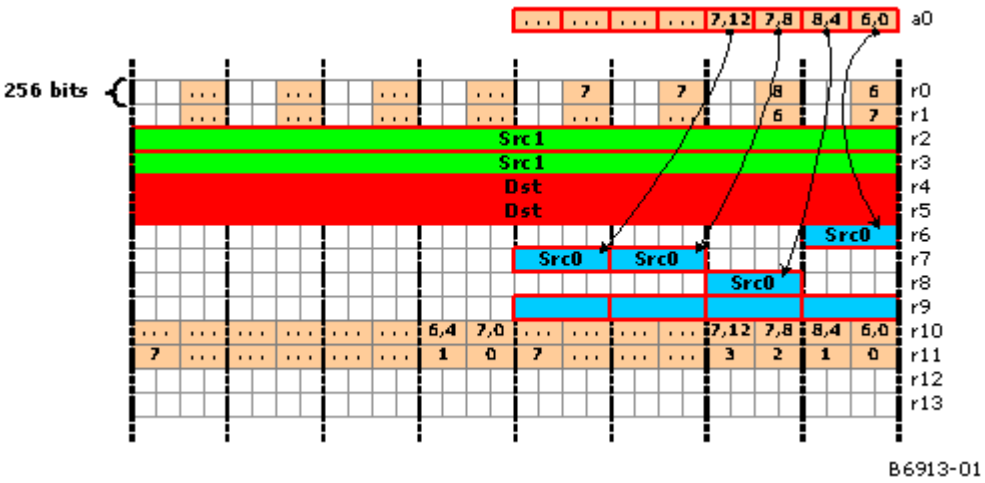
mov (8)a0.0<1>:wr10.0<0;8,1>:w

op (8)r4.0<1>:fr[a0.0]<1,0>:fr2.0<0;8,1>:w// Operate on the first half

mov (8)a0.0<1>:wr10.8<0;8,1>:w// Index values are off by a reg (32b)

op (8)r5.0<1>:fr[a0.0+32]<1,0>:fr3.0<0;8,1>:w// Operate on the second half.

Pixel Shader example using vector immediate.



Without vector immediate support, such translation has to either use a long sequence of scalar instructions which is very inefficient or use a constant load which requires additional constant to be managed in memory.

Supporting OpenGL Vertex Shader Instruction SWZ

When an OpenGL Vertex Shader program is converted to run on GEN in Vertex Pair, i.e. two 4-wide vectors in parallel, the special OpenGL Shader instruction SWZ (Swizzle) needs to be emulated. OpenGL SWZ instruction uses an extended swizzle control field that, in addition to the 4-wide full swizzle control, also includes constant 0 and 1 replacement as well as per channel sign reversal. The later two are not supported by the GEN native instruction. The vector immediate can significantly reduce the overhead of emulating such OpenGL instruction.

Consider an OpenGL Shader instruction in the form of

SWZr1r0.0-zx-1// Expected results: r1.x = 0; r1.y = -r0.z; r1.z = r0.x; r1.w = -1

It can be emulated by the following three GEN instructions.

mul(8)r1.0<1>:fr0.xzxz0x1F111F11:v// Constant vector of (1 -1 1 1 1 -1 1 1)

mov (1)f0.08b'10011001// Set flag & masked out channels y and z

(f0.0)mov(8) r1.0<1>:f 0x000F000F:v// Constant vector of (0 0 0 -1 0 0 0 -1)

In case that only 0, 1, -1 channel replacement is used and there is no signed swizzle, it may be emulated in two GEN instructions. This is illustrated by the following example:

OpenGL:

SWZr1r0.0zx-1// Expected results: r1.x = 0; r1.y = r0.z; r1.z = r0.x; r1.w = -1

GEN:

mov (1)f0.08b'01100110// Set flag and masked out channels x and w

(f0.0)sel (8) r1.0<1>:f r0.yzxy0x000F000F:v// Constant vector of (0 0 0 -1 0 0 0 -1)

Destination Mask for DP4 and Destination Dependency Control

The following example demonstrates the use of destination mask mode of floating point dot-product instruction as well as the use of destination dependency control to improve performance (i.e., avoiding unnecessary thread switch due to possible false dependencies).

Consider a generic DirectX 10 Vertex Shader macro of matrix-vector product that is implemented on GEN in the pair of 4-component vector mode. The DirectX 10 equivalent Shader instructions are as the following.

```
dp4 r5.x r0 r4

dp4 r5.y r1 r4

dp4 r5.z r2 r4

dp4 r5.w r3 r4
```

With destination dependency control, the GEN instructions are as the following. The first instruction in the sequence checks for the destination dependency, but does not clear the dependency bit. The subsequent two instructions would do neither of them. The last instruction avoids checking the destination dependency, but at completion, it clears the destination scoreboard. It ensures that the content of the destination register is coherent, if any of the following instructions uses the same register as source.

```
dp4 (8) r5.0<1>.x:f r0.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr}

dp4 (8) r5.0<1>.y:f r1.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr }

dp4 (8) r5.0<1>.z:f r2.0<4;4,1>:f r4.0<4;4,1>:f {NoDDClr }
```

Just as a comparison, IF GEN DP4 implies reduction at the destination; additional shifted moves are required to achieve the same results. The corresponding codes are as the following. The lower performance due to the additional three move instruction as well as added back-to-back dependencies shows that why we choose to implement the destination channel replication for floating point DP4.

```
dp4 (8) r5.0<1>.y:f r1.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.1<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.z:f r2.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.2<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.w:f r3.0<4;4,1>:f r4.0<4;4,1>:f

mov (1) r5.3<1>:f r8.0<1;1,1>:f

dp4 (8) r5.0<1>.x:f r0.0<4;4,1>:f r4.0<4;4,1>:f
```

Null Register as the Destination

Null register can be used as the destination for most of the instructions. Here are some example usages.

- Null as destination for regular ALU instructions: As all ALU instructions can be configured to update the flag registers using the conditional modifiers, it is not necessary to have a destination register if the programmer only cares about the conditionals of the operation. In that case, a null in the destination operand field saves register space as well as one less dependency checking.
- Null as the destination for SEND/STOR instructions: for the send instruction that only send messages out to an external unit and does not require any return data or feedback, a null in the destination register field signifies the case.
    - One extension of such case is that even though the operation does not have any return values, a return phase with no payload but simply updating the scoreboard flag for a non-null register can provide a signaling mechanism between the thread and the target external unit. One application of this usage is to allow software to manage the coherency of shared memory resources such like the many caches in the system (particularly, valuable for read/write caches).

Use of LINE Instruction

LINE instruction is specifically designed to speed up floating point vector/matrix computation when a program operates in channel serial.

The following example demonstrates how to use LINE instruction to compute Line Equations for DirectX 10 Pixel Shader. In this example, 2 sets of (Cx#, Cy#, Don't Care, C0#) 4-tuple coefficient vectors are stored in registers R1.

R1: Cx0 Cy0 DC Co0 Cx1 Cy1 DC Co1

8 sets of coordinate 2-D vectors (X, Y) are stored in R2 and R3 in the channel serial mode as

R2: X0 X1 … X7

R3: Y0 Y1 … Y7

The objective is to compute the following two line equations for each set of 2D coordinate and store the results in R4 and R5 as

R4: (X0*Cx0 + Y0*Cy0+Co0) … (X7*Cx0 + Y7*Cy0+Co0)

R5: (X0*Cx1 + Y0*Cy1+Co1) … (X7*Cx1 + Y7*Cy1+Co1)

Example LINE Equations

```
//-------------
// Example compute LINE equation in channel serial scenario
//-------------
line (8) acc:f  r1<0;1,0>:f  r2<0;8,1>:f// does acc = X# * Cx0 + Co0
mac (8) r4<1>:f r1.1<0;1,0>:f r3<0;8,1>:f// does r4.# = Y# * Cy0 + acc.#
line (8) acc:f  r1<0;1,0>:f  r2<0;8,1>:f// does acc = X# * Cx0 + Co0
```

mac (8) r4<1>:f r1.1<0;1,0>:f r3<0;8,1>:f// does r4.# = Y# * Cy0 + acc.#

The next example is to compute homogeneous dot product for OpenGL pixel shader running in Channel Serial. In this example, an original OpenGL PS instruction is like

dph R2.x R0 R1

With register remapping, we can store the input coefficient vector R0 in original format in r0, but 8 sets of input coordinate vectors in channel serial format in r2, r3, r4 and r5, and the destination R2.x component in r6.

r0: Cx0 Cy0 Cz0 Co0 DC DC DC DC

r2: X0 X1 … X7

r3: Y0 Y1 … Y7

r4: Z0 Z1 … Z7

r5: W0 W1 … W7

The objective is to compute the following DPH equations and store the results in r6 as

R6: (X0*Cx0+Y0*Cy0+Z0*Cz0+Co0) ... (X7*Cx0+Y7*Cy0+Z7*Cz0+Co0)

Example Homogeneous Dot Product in Channel Serial

//-------------

// Example compute homogeneous dot product in channel serial scenario

//-------------

line (8) acc:f r0<0;1,0>:f r2<0;8,1>:f// does acc = X# * Cx0 + Co0

mac (8) acc:f r0.1<0;1,0>:f r3<0;8,1>:f// does acc.# = Y# * Cy0 + acc.#

mac (8) r6<1>:f r0.2<0;1,0>:f r4<0;8,1>:f// does r6.# = Z# * Cz0 + acc.#

Mask for SEND Instruction

Execution mask (upto 16 bits) for the SEND instruction is transferred to the Shared Function. This provides optimized implementation of DirectX Shader instructions.

Channel Enables for Extended Math Unit

The following example demonstrates how to use the SEND instruction to get service from the Extended Math unit.

Let's consider COS instruction in DirectX 10 in the following form

[([!]p0.{select|any|all})] cos[_sat] dest[.mask], [-]src0[_abs][.swizzle]

For a SIMD4x2 VS implementation with the following register mappings

p0 =>f0.0

src0 =>r0

dest =>r1

The equivalent GEN instruction is as the following

[([!]f0.0.{select|any4h|all4h})] SEND (8) r1[.mask]:f m0 [-][(abs)]r0[.swizzle]:f MATHBOX|COS[|SAT]

If the source swizzle is replication, the message description field can be modified to MATHBOX|COS|SCALAR to take advantage of the fast mode (scalar mode) supported by the Extended Math. The implied move of the SEND instruction is equivalent to the following instruction:

MOV (8) m0[.mask]:f [-][(abs)]r0.0[.swizzle]:f {NoMask}

For a SIMD16 PS implementation, the register mappings are as the followings

p0 =>f0…f3 // in order of R, G, B, A

src0 =>r0,r1; r2,r3; r4,r5; r6,r7

dest =>r8,r9; r10,r11; r12,r13; r14,r15

There are several ways to translate the DirectX instruction, depending on the operand/instruction modifiers present in the DirectX instruction. If predicate is not present and the source swizzle is replication, say, src0.y, which is r2-r3, the translation could be as the following instructions

send (8) r8:f m0 -(abs)r2:f MATHBOX|COS

send (8) r9:f m1 -(abs)r3:f MATHBOX|COS {SecHalf}// use the second half of 8 flag bits

mov (16) r10:fr8:f // All destination color chan's are same

mov (16) r12:fr8:f // MOV is faster than most MathBox func's

mov (16) r14:fr8:f // These MOV's are compressed instructions

Notice that instead of issuing Extended Math messages with the same input data, destination color channel replication is performed by the MOV instructions. This is faster for the thread for most cases as many Extended Math functions consume multiple cycles. This also conserves message bus bandwidth as well as the usage of the shared resource – Extended Math. The destination mask in the DirectX 10 instruction indicates which of the r8 to r15 registers are updated. If the source swizzle is not replication, there will be 8 SEND instructions.

With predication on, if the predication modifier is p0.select, translation is to take the selected flag register f#. The other predication modifiers '.any' and '.all' are translated into '.any4v' and '.all4v', respectively. Notice that with predication on, it is not required to run all 4 pixels in a subspan in the same way, so no need to enforce .any4h/.any4v. The following example shows the instruction with predication (but without .select modifier).

(f0[.any4v|.all4v]) send (8) r8:f m0 -(abs)r2:f MATHBOX|COS

(f0[.any4v|.all4v]) send (8) r9:f m1 -(abs)r3:f MATHBOX|COS {SecHalf}

(f1[.any4v|.all4v]) mov (16) r10:fr8:f // All destination color chan's are same

(f2[.any4v|.all4v]) mov (16) r12:fr8:f // MOV is faster than most MathBox func's

(f3[.any4v|.all4v]) mov (16) r14:fr8:f // These MOV's are compressed instructions

The same instructions works also for predication with select component modifier. We simply replase f0 to f3 above by the selected flag register, say, f1. The modifier of any4h/all4v would also work.

Channel Enables for Scratch Memory

The following example demonstrates how to use the SEND instruction to get service from the Data Port for scratch memory access.

Let's consider general instruction in DirectX 10 that uses scratch memory as a source operand

[([!]p0.{select|any|all})] add dest[.mask], [-]src0[_abs][.swizzle], [-]src1[_abs][.swizzle]

For a SIMD4x2 VS implementation with the following register mappings

p0 =>f0

src0 =>r0

src1 =>s2 / r10

dest =>r1

In this example, the scratch memory offset is provided by an immediate and a GRF register r10 is used as the intermediate GRF location for spill/fill of scratch buffer accesses. This arithmetic instruction is converted into a Data Port read followed by an arithmetic instruction.

mov (8) r3:d r0:d {NoMask}// move scratch base address to be assembled with offset values

mov (1) r3.0:d 2*32 {NoMask}// s2 for vertex 0

mov (1) r3.1:d 2*32+16 {NoMask}// s2 for vertex 1

send (8) r10 m0 r3 DATAPORT|RC|READ_SIMD2

[([!]f0.{sel|any4h|all4h})] add (8) r1[.mask]:f [-][(abs)]r0[.swizzle]:f [-][(abs)]r10[.swizzle]:f

So if scratch register is the source, there is no need to use the channel enable side band. This is also true for channel-serial PS cases.

Now, let's consider the case when a scratch register is the destination of an instruction.

p0 =>f0

src0 =>r0

src1 =>r1

dest =>s2 / r10

We have

add (8) m1:f [-][(abs)]r0[.swizzle]:f [-][(abs)]r1[.swizzle]:f

mov (8) r3:d r0:d {NoMask}// move scratch base address to be assembled with offset values

mov (1) r3.0:d 2*32 {NoMask}// s2 for vertex 0

mov (1) r3.1:d 2*32+16 {NoMask}// s2 for vertex 1

```
[([!]f0.{sel|any4h|all4h})] send (8) null[.mask] m0 r3 DATAPORT|RC|WRITE_SIMD2
```

Notice that with a null as the posted destination register, we are able to transfer the [.mask] over the message channel enables. In many cases for scratch memory assess, a write-with-commit is required, therefore, the posted destination register could be r10.

Now, let's consider the PS case when a scratch register is the destination of an instruction.

p0 =>f0-f4

src0 =>r0-r7

src1 =>r8-r15

dest =>s16-s23 / r16-r23

When predication is not on (or predication with swizzle control on), we have

add (16) m4:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m6:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m8:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

add (16) m10:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (8) r3:d 0x76543210:v {NoMask}// ramp function

mul (16) acc0:d r3:d 16 {NoMask}// ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf}// ramp function

add (16) m2:d acc0:d 2*256 {NoMask}// ramp function

```
send (16) null m1 r3 DATAPORT|RC|WRITE_SIMD16
```

As there is no bit left from the unit specified descriptor field, the 4 bit mask must be put into the header field in m1, which requires at least two more instructions.

Alternatively, or for the case that predication without modifier is on, we can do a read-modify-write.

mov (8) r3:d 0x76543210:v {NoMask}// ramp function

mul (16) acc0:d r3:d 16 {NoMask}// ramp function

add (8) acc0:d acc0:d 64 {NoMask,SecHalf}// ramp function

add (16) m2:d acc0:d 2*256 {NoMask}// ramp function

```
send (16) r16 m1 r3 DATAPORT|RC|READ_SIMD16 // read from scratch
```

// some of the following four instructions may be omitted based on [.mask] field

[([!]f0.{sel|any4v|all4v})] add (16) r16:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r18:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r20:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

[([!]f0.{sel|any4v|all4v})] add (16) r22:f [-][(abs)]r0/2/4/6_BasedOnSwizzle:f [-][(abs)] r8/10/12/14_BasedOnSwizzle:f

mov (16) m4:f r16:f {NoMask}

mov (16) m6:f r18:f {NoMask}

mov (16) m8:f r20:f {NoMask}

mov (16) m10:f r22:f {NoMask}

```
send (16) null m1 null DATAPORT|RC|WRITE_SIMD16 {NoMask}// write back to scratch
```

Flow Control Instructions

Unconditional branches are performed through direct manipulation of the 32-bit IP architectural register. For example:

```
mov (1) IP <memory_address>// jump absoluteadd (1) IP IP <byte_count>// jump relative
```

Note that jump distances are specified in terms of bytes, as opposed to instruction counts in the case of *break*, *halt*, etc. To minimize confusion, an assembler-only instruction 'jmp <inst_count>', where <inst_count> is an immediate term, may be defined which takes an instruction count for a distance. The *jmp* pseudo-opcode can be mapped to an "add (1) ip ip <inst_count> * 16" instruction.

IP is aligned to an 8-byte boundary, thus the 3 LSBs are not maintained in the IP architectural register and should not be relied upon by software.

IP, when used as a source operand, reflects the memory address of the instruction in which it is used. The following are examples illustrating the use of IP:

```
add (1) IP4*16// jumps to HERE_1
add (1) IP0x35// jumps to HERE_1 (4 lsb's don't-care)  <instruction>
 <instruction>

HERE_1:<instruction>HERE_2:<instruction>
 <instruction>
 add (1) IP -2*16// jumps to HERE_2 ...
 add (1) IP 0// infinite loopadd (1) IP 0xF// infinite loop ...
```

Note for Assembler: The if/iff/else/while/break instructions identify relative addresses as the targets of an implicit jump associated with the instruction. These are optional in the assembly syntax as the jitter can determine the location of the matching instruction (e.g. matching endif instruction for a given if instruction).

Execution Masking

This topic is currently under development.

Branching

Example. If / Else / EndIf

```
//-------------
// Example if/else/endif scenario
// "if (r5==r4) ...else ... end-if"
//-------------
...
cmp.e.f0 (8) null r5 r4// does r5 == r4?
(f0) if (8)  HERE_1// "if" part - save then update IMASK;
//   or goto the 'else' if all false
...
...
HERE_1:// now do the 'else' part
else (8) HERE_2// "else" part - invert IMASK
//   or goto the 'endif' if all false
...
...
HERE_2:
endif// "end-if" part – restore IMASK
....// and continue...
```

If it is known that the code has no nested conditionals, a predicate can be used for a lower overhead, more efficient if/else/endif. (One must consider the probability of all channels taking the same branch, and the number of instructions under the if/else blocks as to which conditional method, predicate or mask, is most efficient).

Fast-If

Below is an example of a fast-if instruction. For the 'iff' instruction, only and iff-endif construct is allowed, as opposed to a if-else-endif. Note that the target address for branching if all enabled channels fail is one instruction beyond the endif, as the 'iff' does not push and update the IMask unless the branch is taken for at least one execution channel.

Example Fast If

```
//-------------
// Example – Fast If
```

```
//One instruction overhead conditional
//-------------
...
cmp.e.f0 (8) null r5 r4// any flag update
...
(f0)iff (8) HERE_1// “fast-if” – only pushes IMask;
//   if execution falls through,
//   else go to HERE_1
...
...
endif// “end-if” part – restores IMask
HERE_1:
...// and continue...
```

Cascade Branching

As there is no ‘elseif’ instruction, a C-like cascade branching such as if / elseif / else / endif, can be realized using the basic building blocks of if / else / endif as shown in the following example. Notice that two ‘endif’s’ are required in order to pop the IStack correctly.

Example. If / Elseif / Else / EndIf

```
//-------------
// Example if/elseif/else/endif scenario
// “if (r5==r4) ...elseif (r6>r7) else ... end-if”
//-------------
...
cmp.e.f0 (8) null r5 r4// does r5 == r4?
(f0)if (8) HERE_1// “if” part - save then update IMask;
//   or go to the ‘else’ part if all false
...
...
HERE_1:// now do the ‘else’ part
else (8) HERE_2// “else if” part - invert IMask
//   or go to the ‘else’ part if all false
```

cmp.g.f0 (8) null r6 r7// is r6 > r7?

(f0)if (8) HERE_3// “if” part - save then update IMask;

// or go to the ‘else’ part if all false

...

...

HERE_3:// now do the ‘else’ part

else (8) HERE_4// “else” part - invert IMask

// or go to the ‘end-if’ part if all false

...

...

HERE_4:

endif// “end-if” part – restore IMask for elseif

HERE_2:

endif// “end-if” part – restore IMask for if

....

Compound Branches

Compound branches are supported through the ability logically combine flag registers for each intermediate result.

Example Compound Branch

//-------------

// Example: “if (r0 > r1) OR (r2 <= r3)”

//-------------

...

cmp.g.f0 (8) null r0:d r1:d// r0 > r1?

cmp.le.f1 (8) null r2:d r3:d// r2 <= r3?

or (1) f0:w f0:w f1:w// combine f0 and f1

(f0) if (8) HERE_1// Can now do normal if/else

...

...

HERE_1:endif

...

Example Compound Branch Using 'Any' or 'All'

```
//-------------
// Example: assuming we're doing a channel-serial vector in r0-r3
// We want to know if all components of the vector are > 0x80
//-------------
...
cmp.g.f0 (16)  null r0 0x80// r0 > 0x80?
cmp.g.f1 (16)  null r1 0x80// r1 > 0x80?
cmp.g.f2 (16)  null r2 0x80// r0 > 0x80?
cmp.g.f3 (16)  null r3 0x80// r1 > 0x80?
(f0.all4v) if (16) HERE_1
...
...// code executed only for those channels
...// where per-channel r0,r1,r2,r3 all > 0x80
...
HERE_1:endif
...// and continue...
```

Looping

Due to GEN's SIMD-16 architecture, it must support the case of up to 16 loops running in parallel. These must be handled as independent loops, each with its own loop-exit condition which could occur after a different number of loop iterations. To account for each channel's progress, a 16b loop-mask 'LMask' is defined with 1b associated to each execution channel. This mask keeps track of which channels remain active inside a loop block.

Basic Do-While Loop

Looping illustrates the most basic loop. Two operations must be accomplished before loop entry. (1) Prior to loop entry, there is some subset of enabled channels as dictated by the code sequence prior. In general, the active status of each channel is indicated in the virtual EMask any point in time. These active channels will become the channels over which the loop is run, and LMask must be initialized with the EMask value. (2) Since a given loop may be nested within another loop, the previous LMask & CMask must be saved to the LStack for later restoration upon loop completion. The 'msave' instruction performs both the save and update in a single instruction, and thus all loop-blocks should be fronted with a "msave LStack LMask" and "msave LStack CMask" operation.

Note that the LMask and CMask share the same mask-stack. Thus, CMask must always be a 1's-subset of the LMask for proper stack operation. This is the case if CMask is updated to LMask each pass through the loop (see *Looping*) and through the 'break' instruction updating both masks.

Each pass through the loop, a loop terminating operation must be evaluated and stored in a flag register. This condition must be evaluated on a channel-by-channel basis as exemplified:

cmp.z.f0(8) null r2 d3// any operation that updates a flag

The result of this operation sets a bit per channel in the specified flag register, which is then used in the 'while' instruction. As loops are performed, channels may become disabled as their termination condition is met.

'While' termination is determined on a channel-by-channel basis by the logical AND of corresponding bit positions of AMask, CMask and the specified flag. If the result is '1' the channel remains enabled for the next pass of the loop; if '0' the channel is disabled until loop fall-through. The 'while' instruction causes the LMask to be updated with the latest result of enabled channels. If any channel remains enabled (LMask != ...000b), an additional pass through the loop is made. Once a channel is terminated for the loop operation, it remains terminated until the loop is complete for all channels.

Upon fall through, the 'while' instruction causes the previously saved LMask & CMask to be popped from the LStack, enabling execution on the same subset of channels enabled prior to loop entry (unless a channel had been otherwise terminate inside the loop via 'halt').

Example Basic Loop Construct

```
//-----------------------

//Example: Basic do-while loop structure

//-----------------------

...

do// save L/CMask & update

BEGIN_LOOP:

mov (1) CMask LMask{NoMask}// update CMask for this pass

...

...

<some flag update>

()while (8) BEGIN_LOOP// cond. branch

//   + restores LMask on fall-through

...
```

Do-While Loop with Break

A loop may also be terminated for any channel via the 'break' instruction. The 'break' instruction causes the corresponding bit positions of enabled channels to be cleared in the LMask. If the updated LMask = ...000b, a branch is made to the specified instruction location. An example is shown below in which the 'break' is at the same conditional-nesting level as the terminating 'while'. Its primary value may simply be to support a "do...break.. while (true)" –type structure for a more direct 1:1 translation from higher-level source code.

Example Loop Construct With Non-Nested 'Break'

```
//-------
//Example: While-true loop
//-------
#define BrkCode(i,d)(i « 16) + d
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask LMask{NoMask} // update CMask for this pass
...
<some flag update>
()break (8) BrkCode(0,HERE_1)// Restores LMask when all
// channels complete loop.
...
...
while (8) BEGIN_LOOP// while true
HERE_1:
...
```

A break condition may occur from various levels of nested-ifs. This gives rise to the possibility that a the loop may terminate from within nested 'if's, and due to the jump inherent in the 'break' instruction, the associated 'endif's are not encountered to clean-up the IStack as nesting levels are exited.

Example Loop Construct With 'Break' From Within Nested If's

```
//-------
//Example: General Loop Structure w/ break inside if's
//-------
#define BrkCode(i,d)(i « 16) + d
do// save L/CMask & update
```

```
BEGIN_LOOP:
mov (1) CMask LMask{NoMask} // update CMask for this pass
...
if ...
if ...
if ...
...
()break (8) BrkCode(3,HERE_1)// we're 3 levels deep, so
...
endif
endif
endif
...
()break (8) BrkCode(0,HERE_1)
...
while (8) <flag_spec> BEGIN_LOOP// cond. branch
// + restores C/LMask on fall-through
HERE_1:
```

Do-While Loop with Continue

A continue instruction ‘cont’ is provided skip to the next iteration of the loop. Because not all channels participating in the loop may be enabled at the time this instruction is executed, some channels may require continuation of the loop. A special mask ‘CMask’ is defined which accounts for channels temporarily disabled for the current loop pass.

Since loops may nested, the CMask must be saved and restored around a loop similar to LMask. Since the CMask value within a properly constructed loop is always a subset of the LMask, it can share the LStack for storage, so long as it is pushed after LMask as shown in *Looping*. This save/restore operations are not required if the loop being entered does not have any occurrence of a continue instruction.

Example Do-While with Continue

```
//-------
//Example: General Loop Structure w/ basic break and cont.
//-------
#define ContCode(i,d)(i « 16) + d
do// save L/CMask & update
BEGIN_LOOP:
mov (1) CMask EMask// re-initialize CMask for this pass
...
...
() cont (8) ContCode(0,HERE_1)
...
HERE_1:
()while (8) BEGIN_LOOP// cond. branch
//   + restores C/LMask on fall-through
...
```

Indexed Jump

Example Indexed Jump

```
//-------------
// Code example shows the use of jmpi to perform a case statement
// of any number of options in 3 jumps
//-------------
.default_execution_size 8
...
jmpi r0<0,1,0>// jump relative, based on r0.a.x
// ----- Jump Table ------
jmp HERE_0// redirect for case 0
jmp HERE_1// redirect for case 1
jmp HERE_2// redirect for case 2
jmp HERE_3// redirect for case 3
...
HERE_0:// ... case 0 ...
...
jmp DONE
HERE_1:// ... case 1 ...
...
jmp DONE
HERE_2:// ... case 2 ...
...
jmp DONE
HERE_3:// ... case 3 ...
...
DONE:
...// and continue...
```

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 8: Media VDBOX

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

VDBOX Registers **1**

MFX Architecture **1**

MFX Introduction 1

MFC Overview 2

Sample Algorithmic Flow 3

Synchronization Mechanism 5

Restrictions 6

MFD Overview 7

MFD Memory Interface 11

MFD Codec-Specific Commands 12

MFX State Model 12

MFX Interruptability Model 13

Decoder Input Bitstream Formats 14

AVC Bitstream Formats – DXVA Short 14

AVC Bitstream Formats – DXVA Long 14

VC1 Bitstream Formats – Intel Long 14

MPEG2 Bitstream Formats – DXVA1 14

JPEG Bitstream Formats – Intel 14

Concurrent Multiple Video Stream Decoding Support 15

MFX Codec Commands Summary **15**

MFX Decoder Commands Sequence 20

Examples for AVC 20

Examples for VC1 22

Examples for JPEG 23

MFX Pipe Common Commands **24**

Bitplane Buffer 27

Video Codecs **28**

AVC (H.264) 29

AVC Common Commands 29

AVC Decoder Commands 30

Session Decoder StreamOut Data Structure 30

AVC Encoder PAK Commands....40
Indirect Data Description....40
Unpacked Motion Vector Data Block....40
Packed-Size Motion Vector Data Block....46
Macroblock Level Rate Control....48
Theory of Operation Overview....50
AVC Encoder MBAFF Support....51
MPEG-2....53
MPEG2 Common Commands....53
MPEG2 Decoder Commands....53
Indirect Data Description....53
MPEG2 Encoder PAK Commands....54
PAK Object Inline Data Description – MPEG-2....54
MFX HW Interface and DXVA Conversion....61
Map DXVA to HW BSpec....61
Map HW Bspec to DXVA....64
VC1 Common Commands....66
VC1 Decoder Commands....66
Handling Emulation Bytes....67
VP8....68
MFX_VP8_STATISTICS - Encoder Only....68
VP8 Encoder StreamOut Format....71
VP8 Common Commands....72
VP8 Decoder Commands....72
VP8 Encoder Commands....73
VP8 Bitstream Formats....73
JPEG and MJPEG....74
JPEG Decoder Commands....74
JPEG Encoder Commands....78
More Decoder and Encoder....80
MFD IT Mode Decode Commands....80
Common Indirect IT-COEFF Data Structure....80
Inline Data Description in AVC-IT Mode....81

Indirect Data Format in AVC-IT Mode 87
Inline Data Description in VC1-IT Mode 91
Indirect Data Format in VC1-IT Mode 96
Inline Data Description in MPEG2-IT Mode 97
Indirect Data Format in MPEG2-IT Mode 99
MFX Deblocking Commands 99
MFX Error Handling 99
Encoder StreamOut Mode Data Structure Definition 100
PAK Multi-Pass 102
Driver Usage 103
Programming Reference 103
Monochrome Picture Processing 103
Context Switch 104
PMSI Support 104
Pipeline Flush 104
MMIO Interface 105
Decoder Registers 107
Encoder Registers 108
MMIO Interface 109
Row Store Sizes and Allocations 111

VDBOX Registers

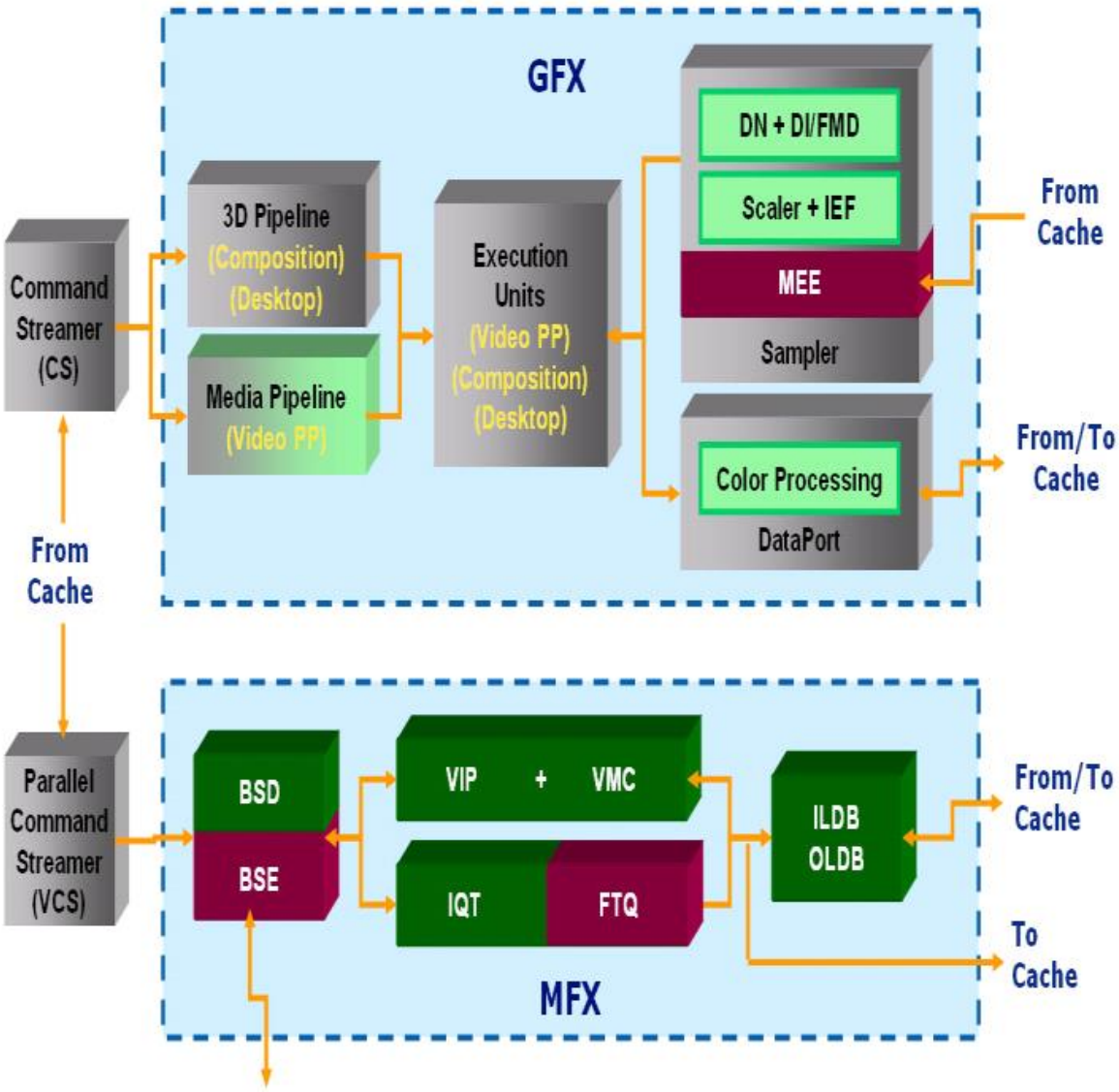
This section describes the VDBOX Command Memory Interface registers.

MFX Architecture

This section and the following sections of Media VDBOX contain the referential documentation on the Multi-Format Codecs, or MFX for those series of chips.

MFX Introduction

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).



MFC Overview

Multi-Format Codec (MFX) Engine is the hardware fixed function pipeline for decode and encoding. It includes multi-format decoding (MFD) and multi-format encoding (MFC).

Note: MFC only supports AVC (H.264).

Many decoding function blocks in MFD such as VIP, VMC, IQT, etc, are also used in encoding mode. Two blocks, FTQ and BSE, are encoding only.

The encoding process is partitioned across host software, the GPE engine, and the MFX engine. The generation of transport layer, sequence layer, picture layer, and slice header layer must be done in the host software. GP hardware is responsible for compressing from Slice Data Layer down to all macro-block and block layers. Specifically, GPE w/ VME acceleration is for motion vector estimation, motion estimation, and code decision. The **VME**(*Video Motion Estimation*) is located next to all image processing units, such as DN (*denoise*) and DI (*deinterlace*) in sampler in GPE. MFX is for final bit packing and reconstructed picture generation.

MFC is operated concurrently with and independently from the GPE (3D/Media) pipeline with a separate command streamer. The two parallel engines have similar command protocol. They can be executed in parallel with different context. For encoding, motion search, MB mode decision, and rate control are performed using GPE pipeline resources.

MFC is implemented to achieve the following objectives:

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
    - Support AVC 4:2:0 Main Profile and High Profile only (8-bit only), up to Level 4.1 resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be encoded. There is no support for Baseline, Extended, or High-10 Profiles.
- Performance requirements with MFX core frequency above 667MHz:
    - Real-time performance with 20% duty cycle or less.
    - Support concurrent decoding of two active HD bitstreams of different formats (for example, one AVC and one VC1 HD bitstream) and one active HD encoding.

As the result of this hardware partitioning, VPP and ENC are always running in GPE, and PAK is what runs exactly in MFC.

PAK – residue packing and entropy coding, including block transformation, quantization, data prediction, bitrate tuning and reference decoding. It delivers final packed bitstream and decoded key-frame reference:

- As the same as ENC, PAK is invoked on a Slice boundary; a single call of VPP can lead to multiple calls for PAK.
- Rate control is inside ENC and PAK only, not in VPP.
- PAK must always perform with reconstructed reference picture.

There is a general dependency of the three operation pipelines. Semaphores are inserted either according to frames or slices. The main CS will also be notified when the decoded reference is ready for the next frame set to be encoded. The detailed discussion will be found in a later section.

Host software is responsible for encoding the transport stream and all the sequence, picture, and slice layer/header in the bit-stream; the MFC system is responsible for compressing from Slice Data Layer down to all macro-block and block layers.

Sample Algorithmic Flow

Assuming all the hardware components are given, there are infinite usage possibilities left with intention for software to decide according to its own application needs depending upon the balanced requirement of coding speed, frame latency, power-consumption, and video quality, and depending upon the usage modes and user preferences (such as low-frame-rate-high-frame-quality vs. high-frame-rate-low-frame-quality).

The last part of this chapter, we illustrate a generic sample to show how a compression algorithm can be implemented to use our hardware.

Step 1. Application or driver initializes the encoder with desired configuration, including speed, quality, targeted bit-rate, input video info, and output format and restrictions.

Step 2. VPP – Application or driver feeds VPP one frame at a time in coded order with specified frame or field type, as well as transcoding informations: motion vectors, coded complexity (i.e. bit size). It will perform denoising and deblocking based on original and targeted bit-rate, and output additional 4 spatial variances and 2 temporal variances for each macroblock as well as the whole frame.

Step 3. ENC – Application or driver feeds ENC one coding slice buffer at a time including all VPP output. The frame level data is accessible to all slices.

a. Encoding setup unit (**ESE**) will set picture level quality parameters (including LUTs, and other costing functions) and set target bit-budget (TBB) and maximal bit-budget (MBB) to each macroblock based on rate-control (**RC**) scheme implemented. For B-frames, it wll also make ME searching mode decision (either Fast, Slow or Uni-directional).
b. Loop over all macroblocks: calculate searching center (**MVP**) perform individual ME and IE (**MEE**). Multi-thread may be designed for HW according to a zigzag order for minimal dependency issue.
c. ENC make microblock level code decision (**CD**) outputs macroblock type, intra-mode, motion-vectors, distortions, as well as TBBs and MBBs.

Step 4. PAK – Application or driver feeds PAK one array of coded macroblocks covering a slice at a time, including all ENC output. Original frame buffer and reconstructed reference frame buffers are also available for PAK to access.

a. PAK may create bitstreams for all sequence, gop, picture, and slice level headers prior the first macroblock.
b. Loop over all macroblocks, accurate prediction block is constructed for either inter- or intra-predictions (**VMC** & **VIP**). If MB distortion is less than some predetermined threshold, for a B slice this step can be skiped as well as the Steps (c)-(e) and jump directly to Step (f); for a key slice the

prediction calculated here will be directly used as the reference thus it jumps to Step (e) after this step.

c. Differencing the predicted block from the original block derives the residue block. Forward transformation and quantization (**FTQ**) is performed. For B slice, it will jump to Step (f) right after. For other types of slice, Steps (d) and (e) can be performed in a thread in parallel with Step (f) and beyond.

d. This is for accurate construction of reference pictures. Inverse quantization and inverse transformation (**IQT**) are performed and added to the predictions to have the decoded blocks.

e. **ILDB** is applied accordingly to the reconstructed blocks.

f. Meanwhile macroblock codes: including its configuration info (types and modes), motion info (motion vectors and reference ids), and residual info (quantized coefficients), are collected for packing (**BSE**) in the following sub-steps:

   i. Code clean-up (in **MPR**). Check and verify Mbtype and Cbps, use Skip or Zero respectively if one can. In principal, when there are equivalent codes, use the simple one.

   ii. Drop dependency (in **MPR**). Calculate relative codes from the absolute codes by associate thm with neighborhood information. All neighborhood correlations are solved in this step.

   iii. Unify symbols (in **SEC**). Translate relative codes into symbols, and table or context indices that are independent of the concept of syntax type.

   iv. Entropy coding (**VLE**) on symbols.

g. Parsing bitstream data in RBSP form (in **VLE**), and output to application or driver.

h. By the end of each picture, write out the accurate actual data size to designate buffer for ENC to access.

Synchronization Mechanism

Encoding of a video stream can be broken down to three major steps (as explained in the previous section):

1. VPP: video-stream pre-processing
2. ENC: encoding, *that is,* code decision of inter-MVs and intra-modes
3. PAK: bit-stream packing
    a. residual calculation, transformation, and quantization
    b. code bit-stream packing
    c. reference generation of keyframes

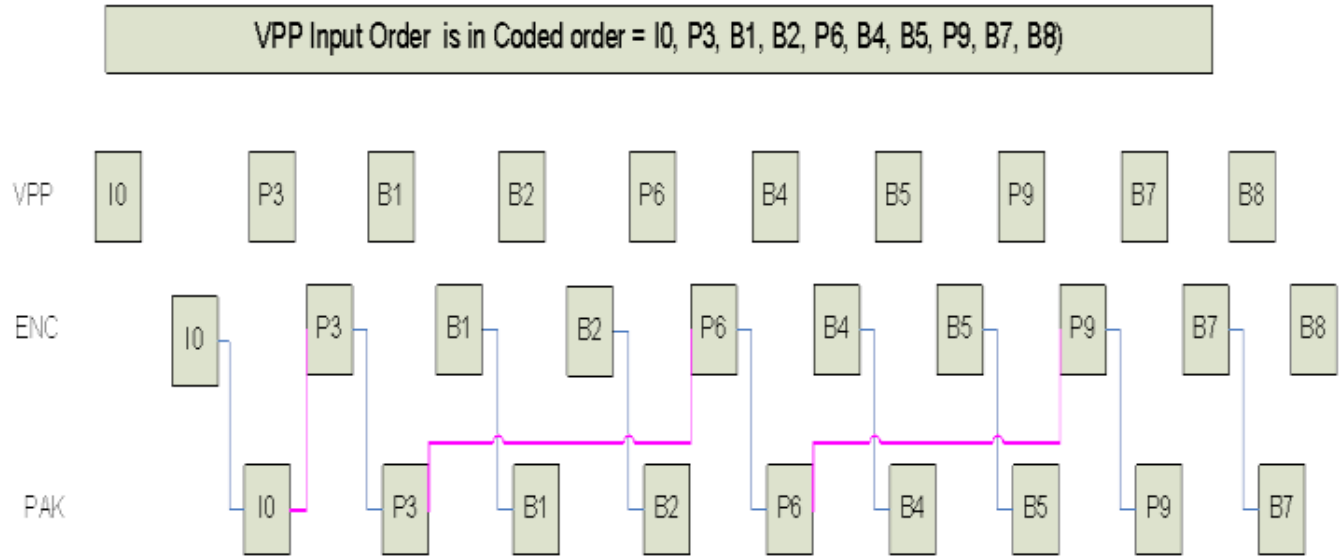
This section describes an architectural solution to map the first two steps in the GFX engine and the last step in the MFX engine. Since this mapping involves two OS-visible engines, managing them in parallel under one application is similar to the solution in earlier generations. Each engine has its own command streamers and has mechanisms to synchronize at required levels as described in the next sub-section.

Above three steps of encoding have dependencies in processing based on

i. functional pipeline order, *i.e.* on a given frame, VPP needs to be performed first, then ENC, then PAK and finally MFD (*Multi-Format Decoding*) for key reference frame generation.
ii. I-frames are key frames for P and B, they have to be first in every pipe-stage.
iii. P-frames are key frames for B frames and therefore P frames are processed first before the dependent B frames
iv. GFX Engine is time slice to work on either VPP or ENC frame as we discussed in the previous chapter.
v. PAK + MFD are executed on the same frame in the MFX engine by macro-block level pipelining within a slice. It should be noted that for the sake of simplicity, an entire frame (potentially multiple slices) are processed in the corresponding engine and no smaller granularity of switching is allowed between the functional pipeline stages.

Three steps of the encoding can be interleaved on two engines in the following way on a frame by frame basis.

Command Stream Synchronization

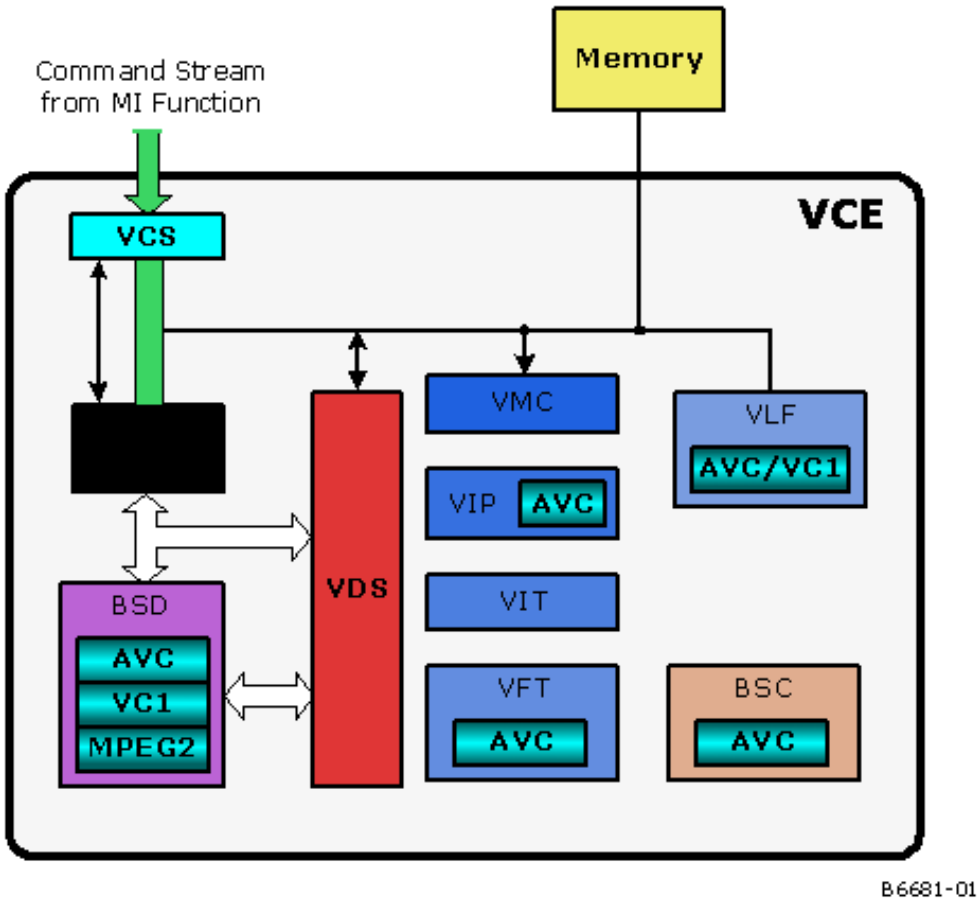


Restrictions

MFC implementation is subject to the following limitations.

- Context switching within MFC and with Graphics Engine occurs only at frame boundary to minimize the amount of information that needs to be tracked and maintained.

MFD Overview



When used for decoding, we also refer to the MFX Engine as the MFD Engine.

The Multi-Format Decoder (MFD) is a hardware fixed function pipeline for decoding the three video codec format and one image compression codec format: AVC (H.264), VC-1, MPEG2, and JPEG.

- Compliant with next generation high definition optical video disc requirements, with sufficient performance headroom:
  - Support AVC 4:2:0 Main and High (8-bit only) Profile only (no support for Baseline, Extended and High-10 Profiles), up to Level 5.1 (max 983,040 MB/s, max 36,864 MB/frame, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
    - Allow a B-picture (frame or field) as a reference picture
  - Support MVC 4:2:0 Stereoscopic Progressive Profile only, up to Level 5.1 (max 983,040 MB/s per view, max 36,864 MB/frame per view, and at most one dimension can reach 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
    - Does not support interlaced video specified in the Stereoscopic Profile

- Support VC1 4:2:0 Simple, Main and Advanced Profiles, up to Level 4 (max 491,520 MB/s and max 16,384 MB/frame; max only one dimension will be at 4K pixel) resolution and up to 40 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded.
  - Allow a B-field as a reference picture only in interlaced field decoding, no other modes.
- Support MPEG2 HD Main Profile (4:2:0), up to High Level (1920x1152 pixels) and up to 80 mbps bitstream. With sufficient duty cycles, higher bit rate contents can also be decoded. No support for SNR and spatial-scalability.
  - Does not support B-picture as a reference picture.
- Support Baseline JPEG with five choma types (4:0:0, 4:1:1, 4:2:2, 4:2:0, and 4:4:4. No support for Extended DCT-based mode, Progressive mode, Loseless mode, nor Hierarchical mode.
  - H/W support 64Kx64K, but Surface State can support only up to 16kx16k

Features	Supported	Unsupported
Coding processes	Baseline sequential mode: • 8-bit pixel precision of source images • loadable 2 AC and 2 DC Huffman tables • 3 loadable quantization matrix for Y, U, V • Interleaved and non-interleaved Scans • Single and multiple Scans	Extended DCT-based mode, Lossless, Hierarchical modes: More than 8 bit pixel resolution, progressive mode, arithmetic coding, 4 AC and 4 DC Huffman tables (extended mode), predictive process (lossless), multiple frames (hierarchical)
Number of image channels	1 for grey image 3 for Y, Cb, Cr color image	4-th channel (usually alpha blending image)
Image resolution	Arbitrary image size up to 16K * 16K	Larger than 16K * 16K (64K * 64K is max. in the JPEG standard)
Chroma subsampling ratio	Chroma 4:0:0 (grey image) Chroma 4:1:1 Chroma 4:2:0 Chroma horizontal 4:2:2 Chroma vertical 4:2:2 Chroma 4:4:4	Any other arbitrary ratio, e.g., 3:1 subsampled chroma
Additional feature (post-processing)	Image rotation: 90/180/270 degrees	

    - H/W does not impose restriction on picture frame aspect ratio, but is bounded by a max 256 MBs (4096 pixels) per dimension programmable at the H/W interface specifications.
        - For example, supporting HD video resolution 1920x1080/60i, 1920x1080/24p, 1280x720/60p
- Performance requirements with MFX core frequency above 1GHz
    - Real-time performance around 10% duty cycle
    - Support concurrently decoding of at least two active HD bitstreams of different formats (For example, one AVC and one VC1 HD bitstream)
- The parsing of transport layer and sequence layer is not performed in this hardware, and is required to be done in the host software.
- The MFD hardware pipeline is operated concurrently with and independently from the Graphics (3D/Media) pipeline with separate command streamer. The two parallel engines are designed with the similar command protocol. They can be executed in parallel with different context.
- Local storages and buffers along the hardware pipeline are kept at minimum. For example, there is no on-die row-store memory. They are resided on the system memory. MFD is designed to hide the memory access latency (in both the row stores and in the motion compensation units) in maximizing its decoding throughput.
- Support the following operating modes:
    - VLD mode - operation starts from entropy decoding of the compressed bit stream (parsing Slice Header and Slice Data Layer in AVC, Picture layer, Slice layer and MB Layer in VC-1, and MB-layer in MPEG2), all the way, to the reconstruction of display picture, including in-loop and out-loop deblocking, if any.
        - Streamout mode - a new feature of the VLD mode in assisting transcoding during decoding. Selected uncompressed data (e.g. per MB MV information) will be sent out to the EU and the ME engine (resided on the Sampler of the 3D Gx Pipeline) for encoding into a different format or for the purpose of transcaling and transrating. In addition, the uncompressed result may continue to be processed by the rest of pipeline as in VLD mode to generate the display picture for transcoding. That is, while intermediate data are streaming out to the memory, the MFD Engine continues its decoding as ususal.
        - For JPEG, only VLD mode is supported (No IT mode). Host software decodes Frame and Scan layers (down to Scan header in the JPEG bit stream syntax) and sends all the corresponding information and Scan payload to the MFD hardware pipeline.
    - IT mode - when host software has already performed all the bit stream parsing of the compressed data and packaging the uncompressed result into a specific format (as a sequence of per-MB record) stored in memory. The hardware pipeline will fetch one MB record at a time and perform the rest of the decoding process as in VLD mode
    - Host software (Application) is responsible for parsing and decoding all the transport and program layers, and all sequence layers. These parameters are passed to Driver and forwarded to H/W as needed through different STATE commands. Host software is also

responsible for separating non-video data (audio, meta and user data) from sending to H/W.

- MFD Engine is only responsible for macro-block and block layers decoding, plus certain level of header decoding. For AVC MFD starts decoding from Slice Header; for VC1, MFD starts decoding from Picture Header, and for MPEG2 decoding starts from MB Layer only.
- For JPEG, MFD is responsible for ECS and block layers decoding.

- Support bitstream formats (compressed video data) for each codec
  - AVC - 2 formats
  - MVC - 2 formats
    - DXVA2 MVC Short Slice Format
    - DXVA2 AVC Long Slice Format Specification (exactly the same as AVC)
  - VC1 - 2 formats
    - Fully compliant to Picture Parameter and Slice Control Parameter interface definition
  - MPEG2
    - MB Layer only, according to DXVA 1 Specification
  - JPEG
    - ECS Layer
- The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.
  - In particularly, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.
- To activate the AVC deblocker logic for incoming uncompressed 4:2:0-only video stream, one can pack the uncompressed video stream to compliant with the IPCM MB data format (including ILDB control information) and feed them into the MFD engine in IT mode. Since the MFD Engine is in IPCM mode, transformation, inter and intra processing are all inactive.

Start Code Detection and removal are done in the CPU, but the Start Code Emulation Prevention Byte is detected and removed by the front end logic in the MFD. The bitstream format for each codec and for each mode is specified in this document.

Codec specific information are based on the following released documents from third parties :

- Draft of Version 4 of H.264/AVC (ITU-T Recommendation H.264 and ISO/IEC 14496-10 (MPEG-4 part 10) Advanced Video Coding); JVT-O205d1.doc; dated 2005-05-30
- Final Draft SMPTE Standard : VC1 Compressed Video Bitstream Format and Decoding Process, SMPTE 421M, dated 2006-1-6; PDF file.
- MPEG2 Recommendation ITU T H.262 (1995 E), ISO/IEC 13818-2: 1995 (E); doc file.
- Digital Compression and Coding of Continuous-tone Still Images, ITU-T Rec. T.81 and ISO/IEC 10918-1: Requirements and guidelines September 18 1992; itu-t81[1].pdf

MFD Memory Interface

The Memory Arbitrator follows the pre-defined arbitration policy (as indicated in the following listing P0 to P11, in which P0 is the highest priority) to select the next memory request to service, then it will perform the TLB translation (translation to physical address in memory), and make the actual request to memory.

The Memory Arbitration unit is also responsible for capturing the return data from memory (read request) and forward it to the appropriate unit along the MFD Engine.

- Read streams: (all 64B requests)
  - Commands for BSD : linear (including indirect data) (P0)
  - Indirect DMA (P1)
  - Row store for BSD: linear (P5)
  - Row store for MPR: linear (P6)
  - MC ref cache fetch : tiled (P2)
  - Intra row store: linear (P9)
  - ILDB row store: linear (P10)
- Write streams: (all 64B requests)
  - Row store write for BSD: linear and can avoid partial writes (P3)
  - Row store write for MPR: linear and can avoid partial writes (P4)
  - Intra row store write: linear and can avoid partial writes (P7)
  - ILDB row store write: linear and can avoid partial writes (P8)
  - Final dest writes: tiled and can potentially be partial, two ways to avoid these partials: 1) either write garbage and buffers are aligned or 2) read-modify writes for dribble end of line cases (P11)

MFD Codec-Specific Commands

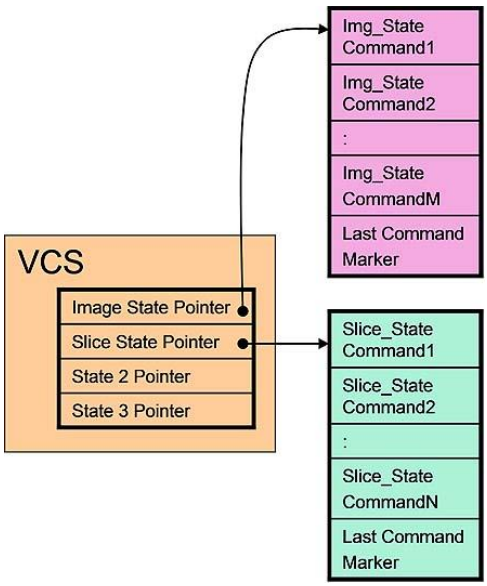
MFD hardware pipeline supports 3 different codec standards : AVC, VC1 and MPEG2. To make the interface flexible, each codec is designed with its own set of commands.

There are two categories of commands for each codec format : one set for VLD mode and one set for IT mode.

MFX State Model

The parallel video engine (PVE) supports two state delivery models: inline state model and indirect state model. For inline state model, the state commands (*_STATE) can be issued in batch buffers or ring buffers directly preceding object commands (*_OBJECT). In the indirect state model, the state commands are not placed in the batch buffers or ring buffers. Instead Indirect State Buffers provide state information (in the form of the above mentioned state commands) for the MFX pipeline. See MFX_STATE_POINTER for more details.

VCS (aka BCS) handles the difference of the two state delivery models. Therefore, the MFX pipeline always sees the state commands in both models. However, MFX hardware supports additional context save/restore of 'dynamic states'. Dynamic states are the internal signals that are persistent. This could be the CABAC context for macroblock encoding.



MFX State Model

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

- In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition.

MFX Interruptability Model

MFX encoding and the encoding pipeline do not support interruption. All operations are frame based. Interrupts can only occur between frames; the driver will submit all the states at the beginning of each frame. Any state kept across frames is in MMIO registers that should be read between frames.

Software submits without any knowledge of where the parser head pointer is located. Also there is a non-deterministic amount of time for the new context to reach the command streamer. However, the state model for the MFX engine requires software to know exactly what state the pipeline is in at all times. This introduces cases where a preemption could occur during or after a state change without software ever knowing the state saved out to memory on the context switch.

Also, preemption is only allowed during the last macroblock in a row. Hardware cannot always perform a context switch when the new context is seen by the hardware. To avoid a switch during an invalid macroblock and to keep the state synchronized with software, there are two commands available that are used. MI_ARB_ON_OFF disables and enables preemption while MFX_WAIT ensures the context switch, if needed, preempts during macroblock execution. Below illustrates an example assuming VC1 VLD mode.

Command Ring/Batch	Notes
MI_ARB_ON_OFF = OFF	Disable preemption
S1	Inline or indirect state cmd 1
S2	Inline or indirect state cmd 2
S3	Inline or indirect state cmd 3
XXXX_OBJECT	Slice
MI_ARB_ON_OFF = ON	Enable preemption
MFX_WAIT	Allow preemption to occur while XXXX_OBJECT executes
MI_ARB_ON_OFF = OFF	Since arbitration is off again, state commands are allowed below
S4	Inline or indirect state cmd 4
S5	Inline or indirect state cmd 5
S6	Inline or indirect state cmd 6
XXXX_OBJECT	Slice
MI_ARB_ON_OFF = ON	Enable preemption
MFX_WAIT	Allow preemption to occur while XXXX_OBJECT executes
MI_ARB_ON_OFF = OFF	Since arbitration is off again, state commands are allowed below

Note that store DW commands may execute inside the preemption enabling window if needed.

Decoder Input Bitstream Formats

AVC Bitstream Formats – DXVA Short

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process.

AVC Bitstream Formats – DXVA Long

Bitstream Buffer Address starts after the 3-byte start code, i.e. starts (and includes) at the NAL Header Byte. This byte must not be included in the Emulation Byte Detection Process. Application will provide the Slice Header Skip Byte count (not including any possible Emulation Prevention Byte).

VC1 Bitstream Formats – Intel Long

Bitstream starts right at the MB layer, with any emulation byte crossing the header and MB layer being removed by application and the data length is adjusted.

MPEG2 Bitstream Formats – DXVA1

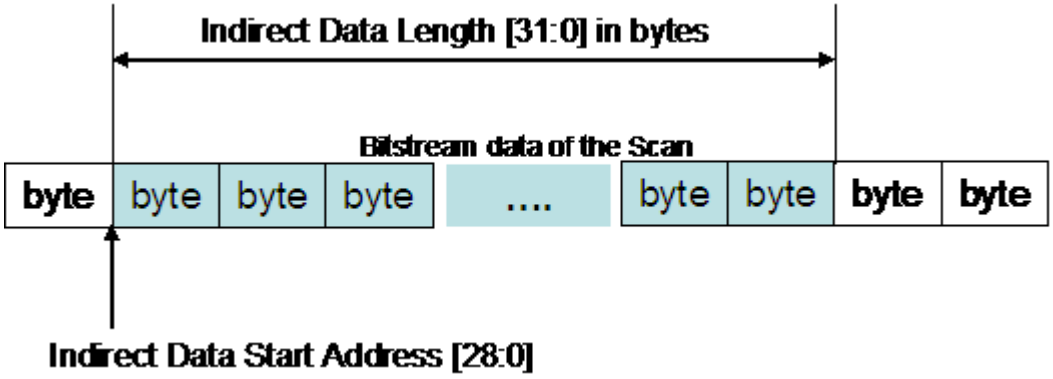
Bitstream buffer starts right at the very first MB data.

JPEG Bitstream Formats – Intel

Bitstream buffer starts right at the very first MCU data of each Scan.

The indirect data start address in MFD_JPEG_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the Scan header. It provides the byte address for the first MCU of the Scan. Different from MFD_MPEG2_BSD_OBJECT command, First MCU Bit Offset does not need to be specified because it is always set to zero.

Indirect data buffer for a Scan



The indirect data length in MFD_JPEG_BSD_OBJECT provides the length in bytes of the bitstream data for the Scan excluding Scan header. It includes the first byte of the first macroblock and the last byte of the last macroblock in the Scan. The Figure illustrates these parameters for a slice data.

Concurrent Multiple Video Stream Decoding Support

The natural place for switching across multiple streams is at the Slice boundary. Each Slice is a self-sustained unit of compressed video data and has no dependency with its neighbors (except for the Deblocking process). In addition, there is no interruptability within a Slice. However, when ILDB is invoked, the processing of some MBs will require neighbour MB information that crosses the Slice boundary. Hence, to limit the buffering requirement, in this version of hardware design, stream switching can only be performed at the picture boundary instead.

MFX Codec Commands Summary

DWord	Bit	Description
0	31:29	**Instruction Type =** GFXPIPE = 3h
	28:16	**3D Instruction Opcode =** PIPELINE_SELECT GFXPIPE[28:27 = 1h, 26:24 = 1h, 23:16 = 04h] (Single DW, Non-pipelined)
	15:1	**Reserved:** MBZ
	0	**Pipeline Select** 0: 3D pipeline is selected 1: Media pipeline is selected

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	Definition Chapter
VC1 State				
2h	5h	0h	VC1_BSD_PIC_STATE	VC1 BSD
2h	5h	1h	Reserved	n/a
2h	5h	2h	Reserved	n/a
2h	5h	3h	VC1_BSD_BUF_BASE_STATE	VC1 BSD
2h	5h	4h	Reserved	n/a
2h	5h	5h-7h	Reserved	n/a
VC1 Object				
2h	5h	8h	VC1_BSD_OBJECT	VC1 BSD
2h	5h	9h-FFh	Reserved	n/a

Pipeline Type (28:27)	Opcode (26:24)	Sub Opcode (23:16)	Command	Definition Chapter	Security
State					
2h	6h	2h-7h	Reserved	N/A	
Object					
2h	6h	9h-FFh	Reserved	N/A	

Note that it is possible for a command to appear in both IMAGE and SLICE state buffer, e.g. QM_STATE for JPEG can be issued at frame level or scan/slice level.

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
	MFX Common	**Common**					
2h	0h	0h	0h	MFX_PIPE_MODE_SELECT	MFX	IMAGE	No
2h	0h	0h	1h	MFX_SURFACE_STATE	MFX	IMAGE	No
2h	0h	0h	2h	MFX_PIPE_BUF_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	3h	MFX_IND_OBJ_BASE_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	4h	MFX_BSP_BUF_BASE_ADDR_STATE	MFX	IMAGE	No
2h	0h	0h	6h	MFX_ STATE_POINTER	MFX	IMAGE	No
2h	0h	0h	7h	MFX_QM_STATE	MFX	IMAGE/SLICE	No
2h	0h	0h	8h	MFX_FQM_STATE	MFX	IMAGE	No
2h	0h	0h	9h	MFX_DBK_OBJECT	MFX	IMAGE	No
2h	0h	0h	A-1Eh	Reserved	n/a	n/a	No
	MFX Common	**Dec**					
2h	0h	1h	0-8h	Reserved	n/a	n/a	n/a
2h	0h	1h	9h	MFD_ IT_OBJECT	MFX	n/a	No
2h	0h	1h	A-1Fh	Reserved	n/a	n/a	n/a
	MFX Common	**Enc**					
2h	0h	2h	0-7Fh	Reserved	n/a	n/a	n/a
2h	0h	2h	8h	MFX_PAK_INSERT_OBJECT	MFX	n/a	No
2h	0h	2h	9h	Reserved	n/a	n/a	n/a
2h	0h	2h	Ah	MFX_STITCH_OBJECT	MFX	n/a	No
2h	0h	2h	B-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Common (State)**					
2h	1h	0h	0h	MFX_AVC_IMG_STATE	MFX	IMAGE	n/a
2h	1h	0h	1h	Reserved	n/a	n/a	n/a
2h	1h	0h	2h	MFX_AVC_DIRECTMODE_	MFX	SLICE	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
				STATE			
2h	1h	0h	3h	MFX_AVC_SLICE_STATE	MFX	SLICE	n/a
2h	1h	0h	4h	MFX_AVC_REF_IDX_STATE	MFX	SLICE	n/a
2h	1h	0h	5h	MFX_AVC_WEIGHTOFFSET_STATE	MFX	SLICE	n/a
2h	1h	0h	9	Reserved	n/a	n/a	n/a
2h	1h	0h	D-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Dec**					
2h	1h	1h	0-5h	Reserved	MFX	n/a	n/a
2h	1h	1h	6h	MFD_AVC_DPB_STATE	MFX	IMAGE	n/a
2h	1h	1h	7h	MFD_AVC_SLICEADDR_OBJECT	MFX	n/a	n/a
2h	1h	1h	8h	MFD_AVC_BSD_OBJECT	MFX	n/a	No
2h	1h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Enc**					
2h	1h	2h	0-8h	Reserved	n/a	n/a	n/a
2h	1h	2h	9h	MFC_AVC_PAK_OBJECT	MFX	n/a	No
2h	1h	2h	A-1Fh	Reserved	n/a	n/a	n/a
	AVC/ MVC	**Extension**					
	VC1	**Common (State)**					
2h	2h	0h	0h	Reserved	n/a	n/a	n/a
2h	2h	0h	1h	MFX_VC1_PRED_PIPE_STATE	MFX	IMAGE	n/a
2h	2h	0h	2h	MFX_VC1_DIRECTMODE_STATE	MFX	SLICE	n/a
2h	2h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	VC1	**Dec**					
2h	2h	1h	0h	MFD_VC1_SHORT_PIC_STATE	MFX	IMAGE	n/a
2h	2h	1h	1h	MFD_VC1_LONG_PIC_STATE	MFX	IMAGE	n/a
2h	2h	1h	2-7h	Reserved	n/a	n/a	n/a
2h	2h	1h	8h	MFD_VC1_BSD_OBJECT	MFX	n/a	No

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	2h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	VC1	**Enc**					
2h	2h	2h	0-1Fh	Reserved	n/a	n/a	n/a
	MPEG2	**Common (State)**					
2h	3h	0h	0h	MFX_MPEG2_PIC_STATE	MFX	IMAGE	n/a
2h	3h	0h	1-1Fh	Reserved	n/a	n/a	n/a
	MPEG2	**Dec**					
2h	3h	1h	1-7h	Reserved	n/a	n/a	n/a
2h	3h	1h	8h	MFD_MPEG2_BSD_OBJECT	MFX	n/a	No
2h	3h	1h	9-1Fh	Reserved	n/a	n/a	n/a
	MPEG2	**Enc**					
2h	3h	2h	0-2h	Reserved	n/a	n/a	n/a
2h	3h	2h	3h	MFC_MPEG2_PAK_OBJECT			
2h	3h	2h	3-8h	Reserved			
2h	3h	2h	9h	MFC_MPEG2_SLICEGROUP_STATE			
2h	3h	2h	A-1Fh	Reserved			
	VP8	**Common (State)**					
2h	4h	0h	0h	MFX_VP8_PIC_STATE	MFX	IMAGE	n/a
	VP8	**Dec**					
2h	4h	1h	8h	MFD_VP8_BSD_OBJECT	MFX	IMAGE	No
	VP8	**Enc**					
2h	4h	2h		Reserved			
	JPEG	**Common**					
2h	7h	0h	0h	MFX_JPEG_PIC_STATE	MFX	IMAGE	No
2h	7h	0h	1h	Reserved	n/a	n/a	n/a
2h	7h	0h	2h	MFX_JPEG_HUFF_TABLE_STATE	MFX	IMAGE	No
2h	7h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	JPEG	**Common**					
2h	7h	0h	0h	MFX_JPEG_PIC_STATE	MFX	IMAGE	No
2h	7h	0h	1h	Reserved	n/a	n/a	n/a

Pipeline Type (28:27)	Opcode (26:24)	SubopA (23:21)	SubopB (20:16)	Command	Chapter	Recommended Indirect State Pointer Map	Interruptable?
2h	7h	0h	2h	MFX_JPEG_HUFF_TABLE_STATE	MFX	IMAGE	No
2h	7h	0h	3-1Fh	Reserved	n/a	n/a	n/a
	JPEG	**Dec**					
2h	7h	1h	1-7h	Reserved	MFX	n/a	n/a
2h	7h	1h	8h	MFD_JPEG_BSD_OBJECT	MFX	MCU	No
2h	7h	1h	9-1Fh	Reserved	MFX	n/a	n/a
	JPEG	**Enc**					
2h	7h	2h	0-1Fh	Reserved	MFX	n/a	n/a

MMIO Space Registers

Range Start	Range End	Unit owner
00002000	00002FFF	Render/Generic Media Engine
00004000	00004FFF	Render/Generic Media Graphics Memory Arbiter
00005000	0000517F	
00006000	00007FFF	Reserved
00012000	000123FF	MFX Control Engine (Video Command Streamer)
00012400	00012FFF	Media Units (VIN unit)
00014000	00014FFF	MFX Memory Arbiter
00022000	00022FFF	Blitter Engine
00024000	00024FFF	Blitter Memory Arbiter
00030000	0003FFFF	Reserved
00100000	00107FFF	Fence Registers
00140000	0017FFFF	MCHBAR (SA)

Memory Interface Command Map

04h Opcode (28:23)	MI_FLUSH

MFX Decoder Commands Sequence

The MFX codec is designed to be a stateless engine, that it does not retain any history of settings (states) for the encoding/decoding process of a picture. Hence, driver must issue the full set of MFX picture state command sequence prior to process each new picture. In addition, driver must issue the full set of Slice state command sequence prior to process a slice.

In particular, RC6 always happens between frame boundaries. So at the beginning of every frame, all state information needs to be programmed. There is no state information as part of media context definition

Examples for AVC

The following gives a sample command sequence programmed by a driver

a) For Intel or DXVA2 AVC Long Slice Bitstream Format

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_QM_STATE

VLD mode: MFX_AVC_PICID_STATE

MFX_AVC_IMG_STATE

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_REF_IDX_STATE

MFX_AVC_WEIGHTOFFSET_STATE

MFX_AVC_SLICE_STATE

VLD mode: MFD_AVC_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 AVC Short Slice Bitstream Format (for VLD mode only)

MFX_PIPE_MODE_SELECT

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFD_AVC_DPB_STATE

VLD mode: MFX_AVC_PICID_STATE

MFX_AVC_IMG_STATE

MFX_QM_STATE

MFX_AVC_DIRECTMODE_STATE

VLD mode : MFD_AVC_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

VLD mode : MFD_AVC_BSD_SLICEADDR_OBJECT

VLD mode: MFD_AVC_BSD_OBJECT

... repeat these four commands N-1 times for a N-slice picture

VLD mode: MFD_AVC_BSD_OBJECT (for the last slice of the picture)

MI_FLUSH

Examples for VC1

The following gives a sample command sequence programmed by a driver

a) For Intel Proprietary Long Bitstream Format

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_LONG_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

IT mode: MFD_IT_OBJECT

MI_FLUSH

b) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only)

MFX_VC1_DIRECTMODE_STATE

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_SHORT_PIC_STATE

VLD mode: MFD_VC1_BSD_OBJECT

MI_FLUSH

c) For DXVA2 VC1 Compliant Bitstream Format (for VLD mode only), and field pair picture

Batch buffer for top-field

states....

Slice_objs...

MI_flush

store register immediate (if VC1 short format with interlaced field pic)

MI_flush

Batch buffer for bottom field

load register immediate (if VC1 short format with interlaced field pic)

MI_flush

states....

Slice_objs...

MI_flush

Examples for JPEG

The following gives a sample command sequence programmed by a driver

Programmed once at the start of decoding

MFX_PIPE_MODE_SELECT

MFX_PIPE_SURFACE_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_JPEG_PIC_STATE

Programmed at the start of Frame or Scan (These commands can be sent multiple times either before MFX_JPEG_PIC_STATE or before MFD_JPEG_BSD_OBJECT)

MFX_JPEG_HUFF_TABLE

MFX_QM_STATE

Programmed per Scan (These commands can be sent multiple times depending on each bit stream)

MFD_JPEG_ BSD_OBJECT

MI_FLUSH

MFX Pipe Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_STATE_POINTER

MFX_PIPE_MODE_SELECT

The Encoder Pipeline Modes of Operation (Per Frame):

1. PAK Mode: VCS-command driven, setup by driver. Like the IT mode of decoder, it is executed on a per-MB basis. Hence, each PAK Object command corresponds to coding of only one MB.

a. Normal Mode (including transcoding): receive per-MB control and data (MV, mb_type, cbp, etc.). It generates the output compressed bitstream as well as the reconstructed reference pictures, one MB at a time, for later use.

b. Encoder StreamOut Mode: to provide per-MB, per-Slice and per-Frame coding result and information (statistics) to the Host, Video Preprocessing Unit and ENC Unit to enhance their operations.

The Decoder Pipeline Modes of Operation (Per Frame):

1. VLD Mode: The output from the BSD (weight&offset/coeff/motion vectors record) can be sent in part (as specified) and to the remaining fixed function hardware pipeline to complete the decoding processing. The driver specifies through MFD commands of what to send out from the BSD unit and where to send the BSD output.

a. For transcoding (including transrating and transcaling), part of the BSD output (a series of per-MB record) can be sent to memory for further processing to encode into a difference output format. This function is named as StreamOut. When StreamOut is active, not all MB information needs to be sent, only MVs and selective MB coding information.

2. IT Mode: In this mode, the BSD is not invoked. Instead host performs all the bitstream decoding and parsing; and the result are saved into memory in a specific per-MB record format. The MFD Engine VCS reads in these records one at time and finish the rest of the decoding (IT, MC, IntraPred and ILDB).

a. MB information is organized into two indirect data buffers, one for MVs and one for residue coefficients. As such, two indirect base address pointers are defined.

Programming Restriction:

- Software must ensure the current pipeline is flushed via an MI_FLUSH prior to the execution of MFX_PIPE_MODE_SELECT in switching the MFX Engine to encode/decode a different codec format (AVC, VC1 or MPEG2).
- MFX_PIPE_MODE_SELECT is issued per picture (frame or field).

MFX_SURFACE_STATE

MFX_PIPE_BUF_ADDR_STATE

MFX_IND_OBJ_BASE_ADDR_STATE

MFX_BSP_BUF_BASE_ADDR_STATE

MFX_PAK_INSERT_OBJECT

MFX_STITCH_OBJECT

MFX_QM_STATE

Bits	31:24	23:16	15:8	7:0	Project
Dword 1	QuantMatrix[0][3]	QuantMatrix[0][2]	QuantMatrix[0][1]	QuantMatrix[0][0]	CHV
Dword 2	QuantMatrix[0][7]	QuantMatrix[0][6]	QuantMatrix[0][5]	QuantMatrix[0][4]	CHV
Dword 3	QuantMatrix[1][3]	QuantMatrix[1][2]	QuantMatrix[1][1]	QuantMatrix[1][0]	CHV
...	...	...	...	...	CHV
Dword 16	QuantMatrix[7][7]	QuantMatrix[7][6]	QuantMatrix[7][5]	QuantMatrix[7][4]	CHV

MFX_FQM_STATE

This is a frame-level state. Reciprocal Scaling Lists are always sent from the driver regardless whether they are specified by an application or the default/flat lists are being used. This is done to save the ROM (to store the default matrices) inside the PAK Subsystem. Hence, the driver is responsible for determining the final set of scaling lists to be used for encoding the current slice, based on the AVC Spec (Fall-Back Rules A and B). For encoding, there is no need to send the qm_list_flags[i], i=0 to7 and qm_present_flag to the PAK, since Scaling Lists syntax elements are encoded above Slice Data Layer.

FQM Reciprocal Scaling Lists elements are 16-bit each, conceptually equal to 1/ScaleValue. QM matrix elements are 8-bit each, equal to ScaleValue. However, in AVC spec., the Reciprocal Scaling Lists elements are not exactly equal to one-over of the corresponding Scaling Lists elements. The numbers are adjusted to simplify hardware implementation.

For all the description below, a scaling list set contains 6 4x4 scaling lists (or forward scaling lists) and 2 8x8 scaling lists (or forward scaling lists).

In MFX PAK mode, PAK needs both forward Q scaling lists and IQ scaling lists. The IQ scaling lists are sent as in MFD in raster scan order as shown in MFX_AVC_QM_STATE. But the Forward Q scaling lists are sent in transport form, i.e. column-wise raster order (column-by-column) to simplify the H/W.

Precisely, if the reciprocal forward scaling matrix is F[4][4], then the 16 word of the matrix will be set as the following:

	bits 0-15	bits 16-31	Project
DW0	F[0][0]	F[1][0]	All
DW1	F[2][0]	F[3][0]	All
DW2	F[0][1]	F[1][1]	All
DW3	F[2][1]	F[3][1]	All
DW4	F[0][2]	F[1][2]	All
DW5	F[2][2]	F[3][2]	All
DW6	F[0][3]	F[1][3]	All
DW7	F[2][3]	F[3][3]	All

For JPEG encoder, 16-bit precision is used for each element 1/QM matrix. The 32 DWords are used for 64 QM elements with the following data structure:

	Bits 15:0	Bits 31:16	Project
DWord1	1/QM[0][0]	1/QM[1][0]	CHV
DWord2	1/QM[2][0]	1/QM[3][0]	CHV
DWord3	1/QM[4][0]	1/QM[5][0]	CHV
DWord4	1/QM[6][0]	1/QM[7][0]	CHV
DWord5	1/QM[0][1]	1/QM[1][1]	CHV
DWord6	1/QM[2][1]	1/QM[3][1]	CHV
DWord7	1/QM[4][1]	1/QM[5][1]	CHV
DWord8	1/QM[6][1]	1/QM[7][1]	CHV
...			CHV
DWord31	1/QM[4][7]	1/QM[5][7]	CHV
DWord32	1/QM[6][7]	1/QM[7][7]	CHV

Bitplane Buffer

Bitplane coding is used in seven different cases in VC-1, although not all the seven syntax elements are present in the same picture header at the same time. The following list shows which syntax elements are coded as bitplanes in each picture header:

Progressive I and BI picture headers in AP: ACPRED, OVERFLAGS
Field interlace I and BI picture headers in AP: ACPRED, OVERFLAGS
Frame interlace I and BI picture headers in AP: FIELDTX, ACPRED, OVERFLAGS

Frame interlace P picture headers in AP: SKIPMB
Progressive P picture headers in SP and MP: MVTYPEMB, SKIPMB
Progressive P picture headers in AP: MVTYPEMB, SKIPMB

Field interlace B picture headers in AP: FORWARDMB
Frame interlace B picture headers in AP: DIRECTMB, SKIPMB
Progressive B picture headers in AP: DIRECTMB, SKIPMB
Progressive B picture headers in MP: DIRECTMB, SKIPMB

There are also seven different modes of coding the bitplane information. Except when the bitplane is coded in raw mode, the bitplane is decoded by the host and provided to the hardware in the bitplane buffer.

Since at most three bitplanes are encoded in any picture header, instead of using a complete byte for signaling the values of all the seven possible bitplanes for each MB, a more efficient approach is used with each byte divided in two nibbles and each nibble carries the data of up to four bitplanes for one MB.

PictureType	Bits 3, 7	Bit 2, 6	Bits 1, 5	Bits 0, 4
I or BI	0	OVERFLAGS	ACPRED	FIELDTX
P	0	MVTYPEMB	SKIPMB	0
B	0	FORWARDMB	SKIPMB	DIRECTMB

The bytes containing the above defined nibbles are stored in the bitplane buffer in raster scan order. The bitplane buffer is a linear buffer with a buffer pitch (as defined by Bitplane Buffer Pitch field in VC1_BSD_PIC_STATE command). When the number of macroblock in a row is odd, the last byte of the row containing the last macroblock in bits 0-3. The first macroblock of the next row starts at the next pitch offset from the first macroblock of the current row.

The bitplane buffer structure must be sent once per picture only if there is one or more syntax elements coded as bitplanes in the picture header.

Video Codecs

The following sections contain the various registers for video codec support. Specifically, the codec types supported are:

Supported Codec Types
Advanced Video Coding (AVC)/ H.264/MPEG-4 Part 10 (MVC)
MPEG-2 (H.222/H.262) — Used in Digital Video Broadcast and DVDs
VC1 — SMPTE 421M, known informally as VC-1
VP8 — Video compression format
JPEG and MJPEG — A video format in which video gram or interlaced field of a digital video sequence is compressed separately as a JPEG image
Other Codec Functions

Internal Media Rowstore table – An internal Media Rowstore Storage is added to reduce memory read/write to save power. If the internal Media Rowstore exists, driver should use the storage as the following table indicates.

AVC/VC1/MPEG2/JPEG/VP8 Decoder/Encoder: (CHV, ,)

[BSD is bitstream decoder rowstore; MPR is Motion Prediction rowstore; IP is Intra Prediction rowstore; VLF is loop filter rowstore; VDE is VDENC rowstore]

Codec	Mode	Frame Width	BSD	MPR	IP	VLF	VDE	BSD Addr	MPR Addr	IP Addr	VLF ADDR	VDE ADDR
AVC	Frame/Field	< 2k	Y	Y	Y	Y	N	0	128	256	384	N/A
		2k -> 3k	Y	Y	Y	N	N	0	192	384	N/A	N/A
		3k -> 4k	Y	Y	N	N	N	0	256	N/A	N/A	N/A
	Mbaff	< 2k	Y	Y	Y	N	N	0	256	512	N/A	N/A
		2k -> 4k	Y	N	N	N	N	0	N/A	N/A	N/A	N/A
AVC	Frame/Field	< 2k	Y	N	Y	Y	N	0	N/A	256	384	N/A
		2k -> 3k	Y	N	Y	N	N	0	N/A	384	N/A	N/A
		3k -> 4k	Y	N	Y	N	N	0	N/A	256	N/A	N/A
	Mbaff	< 2k	Y	N	Y	N	N	0	N/A	512	N/A	N/A
		2k -> 4k	Y	N	N	N	N	0	N/A	N/A	N/A	N/A
JPEG Dec/Enc			N	N	N	N	N	N/A	N/A	N/A	N/A	N/A
VP8 Dec	Frame	< 2k	Y	N	Y	Y	N	0	N/A	256	384	N/A
		2k -> 3k	Y	N	Y	N	N	0	N/A	384	N/A	N/A
		3k -> 4k	Y	N	Y	N	N	0	N/A	256	N/A	N/A
VP8 Enc	Frame		N	N	N	N	N	N/A	N/A	N/A	N/A	N/A
MPEG2			N	N	N	N	N	N/A	N/A	N/A	N/A	N/A
VC1 Dec			Y	N	N	N	N	N/A	N/A	N/A	N/A	N/A

AVC (H.264)

AVC Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_AVC_IMG_STATE

A new command is added to support MPEG transport stream encapsulation of AVC bitstream in Encoder mode. This command should be used only when MPEG transport stream is needed.

MFX_AVC_DIRECTMODE_STATE

MFX_AVC_SLICE_STATE

MFX_AVC_REF_IDX_STATE

MFX_AVC_WEIGHTOFFSET_STATE

AVC Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_AVC_DPB_STATE

NOTE modified from DXVA2 – The values in RefFrameList and UsedForReference_Flag are the primary means by which the H/W can determine whether the corresponding entries in RefFrameList, POCList, LTSTFrameNumList, and Non-ExistingFrame_Flag should be considered valid for use in the decoding process of the current picture or not. When RefFrameList[i] is marked to be invalid, the values of POCList[i][0], POCList[i][1], LTSTFrameNumList[i], UsedForReference_Flag[i], and Non-ExistingFrame_Flag[i] must all be equal to 0. When UsedForReference_Flag[i] = 0, the value of RefFrameList[i] must be marked invalid.

MFD_AVC_SLICEADDR

MFD_AVC_BSD_OBJECT

Inline Data Description for MFD_AVC_BSD_Object

MFD_AVC_PICID_STATE

NOTE 1: In AVC short format, PictureIDList has one-to-one corresponding to LongTermFrame_Flag list, Non-ExistingFrame_flag list, UsedForReference_Flag list, FrameNumList list in MFD_AVC_DPB_STATE.

NOTE 2: PictureIDList is only used to identify reference picture across frames. Hardware will convert the picture in the RefFrameList to PictureID before writing out DMV data and convert back to RefFrameList Index after reading out DMV data. The reference pictures and their orders in the RefFrameList can be changed across frames.

Session Decoder StreamOut Data Structure

When StreamOut is enabled, per MB intermediated decoded data (MVs, mb_type, MB qp, etc.) are sent to the memory in a fixed record format (and of fixed size). The per-MB records must be written in a strict raster order and with no gap (i.e. every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

A StreamOut Data record format is detailed as follows:

DWord	Bit	Description
	23	**Reserved MBZ**
	22-20	**EdgeFilterFlag** (AVC) / **OverlapSmoothFilter** (VC1)
	19:17	**CodedPatternDC** (for AVC only, **111b** for others) The field indicates whether DC coefficients are sent. 1 bit each for Y, U and V.
	16	**Reserved MBZ**
	15	**Transform8x8Flag** When it is set to 0, the current MB uses 4x4 transform. When it is set to 1, the current MB uses 8x8 transform. The transform_szie_8x8_flag syntax element, if present in the output bitstream, is the same as this field. However, whether transform_szie_8x8_flag is present or not in the output bitstream depends on several conditions: This field is only allowed to be set to 1 for two conditions: It must be 1 if **IntraMbFlag** = INTRA and **IntraMbMode** = INTRA_8x8 It may be 1 if **IntraMbFlag** = INTER and there is no sub partition size less than 8x8 Otherwise, this field must be set to 0. 0: 4x4 integer transform 1: 8x8 integer transform
	14	**MbFieldFlag** This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. Same as the mb_field_decoding_flag syntax element in AVC spec. 0 = Frame macroblock 1 = Field macroblock
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) **1: INTRA (intra macroblock)**
	12:8	**MbType5Bits** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows AVC encoding for inter and intra macroblocks.

DWord	Bit	Description
	7	**MbPolarity** FieldMB Polarity - vctrl_vld_top_field AVC
	6	**Reserved MBZ**
	5:4	**IntraMbMode** This field is provided to carry information partially overlapped with MbType. This field is only valid if **IntraMbFlag** = INTRA, otherwise, it is ignored by hardware.
	3	**Reserved MBZ**
	2	**MbSkipFlag** Reserved MBZ (DXVA Encoder). HW (VDSunit) doesn't have skip MB info. It sets to 1 if any of the sub-blocks is inter, uses predicted MVs, and skips sending MVs explicitly in the code stream. Currently H/W can provide this flag and is defaulted to 0 always.
	1:0	**InterMbMode** This field is provided to carry redundant information as that in MbType. It also carries additional information such as skip. This field is only valid if **IntraMbFlag** =INTER, otherwise, it is ignored by hardware.
1	31:16	**MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
	15:0	**MbXCnt (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
2	31	**Conceal MB Flag.** This field specifies if the current MB is a conceal MB, use in AVC/VC1/MPEG2 mode.
	30	**Last MB of the Slice Flag.** This field indicate the current MB is a last MB of the slice. Use in AVC/VC1/MPEG2 mode.
	29:24	**Reserved**
	23:20	**CbpAcV** 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).
	19:16	**CbpAcU** 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).

DWord	Bit	Description
	15:0	**CbpAcY** 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding). Bit15=Y0Sub0, Bit0=Y3Sub3
3	31:28 AVC	**Skip8x8Pattern** (AVC) This field indicates whether each of the four 8x8 sub macroblocks is using the predicted MVs and will not be explicitly coded in the bitstream (the sub macroblock will be coded as direct mode). It contains four 1-bit subfields, corresponding to the 4 sub macroblocks in sequential order. The whole macroblock may be actually coded as B_Direct_16x16 or B_Skip, according to the macroblock type conversion rules described in a later sub section. This field is only valid for a B slice. It is ignored by hardware for a P slice. Hardware also ignores this field for an intra macroblock. 0 in a bit – Corresponding MVs are sent in the bitstream 1 in a bit – Corresponding MVs are not sent in the bitstream
	27:25	**Reserved**
	24:16	**NzCoefCountMB** – all coded coefficients input including AC/DC blocks in current MB. Range 0 to 384 (9 bits)
	15:8	**[CHV] MbClock16** – MB compute clocks in 16-clock unit.
	7	**mbz** (AVC) **/ QScaleType** (MPEG2)
	6:0	**QpPrimeY** (AVC) **/ QScaleCode** (MPEG2) The luma quantization index. This is the per-MB QP value specified for the current MB.
4 to 6	31:0 Each	For intra macroblocks, definition of these fields are specified in 1 For inter macroblocks, definition of these fields are specified in 2
7	31:24	**Reserved**
	23:20	**MvFieldSelect** (Ref polarity top or bottom bits) for VC1 and MPEG2 vcp_vds_mvdataR[162:159] VC1 vmd_vds_mt_vert_fld_selR[3:0] MPEG2
	19:12	**Reserved**
	11:10	**SubBlockCodeType V** (If 8x8, 8x4, 4x8, 4x4 type)

DWord	Bit	Description
	9:8	**SubBlockCodeType U** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	7:6	**SubBlockCodeType Y3** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	5:4	**SubBlockCodeType Y2** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	3:2	**SubBlockCodeType Y1** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
	1:0	**SubBlockCodeType Y0** (specifies 8x8, 8x4, 4x8, 4x4 type) VC1
Inter Cases:		
8	31:16	**MvFwd[0].y** – y-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock
	15:0	**MvFwd[0].x** – x-component of the forward motion vector of the 1st 8x8 or 1st 4x4 subblock
9	31:0	**MvBck[0]** – the backward motion vector of the 1st 8x8 or 1st 4x4 subblock
10	31:0	**MvFwd[1]** – the forward motion vector of the 2nd 8x8 or 4th 4x4 subblock
11	31:0	**MvBck[1]** – the backward motion vector of the 2nd 8x8 or 4th 4x4 subblock
12	31:0	**MvFwd[2]** – the forward motion vector of the 3rd 8x8 or 8th 4x4 subblock
13	31:0	**MvBck[2]** – the backward motion vector of the 3rd 8x8 or 8th 4x4 subblock
14	31:0	**MvFwd[3]** – the forward motion vector of the 4th 8x8 or 12th 4x4 subblock
15	31:0	**MvBck[3]** – the backward motion vector of the 4th 8x8 or 12th 4x4 subblock
Intra Cases:		
8 to 15	31:0	**Reserved MBZ**

The inline data content of Dwords 4 to 6 is defined either for intra prediction or for inter prediction, but not both.

Inline data subfields for an Intra Macroblock

DWord	Bit	Description
4	31:16	**LumaIndraPredModes[1]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[0]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block, four 8x8 block or one intra16x16 of a MB. 4-bit per 4x4 sub-block (Transform8x8Flag=0, Mbtype=0 and intraMbFlag=1) or 8x8 block

DWord	Bit	Description
		(Transform8x8Flag=1, Mbtype=0, MbFlag=1), since there are 9 intra modes. 4-bit for intra16x16 MB (Transform8x8Flag=0, Mbtype=1 to 24 and intraMbFlag=1), but only the LSBit[1:0] is valid, since there are only 4 intra modes. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
5 AVC INTRA	31:16	**LumaIndraPredModes[3]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment table later in this section. VC1: MBZ. MPEG2: MBZ.
	15:0	**LumaIndraPredModes[2]** Specifies the Luma Intra Prediction mode for four 4x4 sub-block of a MB, 4-bit each. AVC: See the bit assignment later in this section. VC1: MBZ. MPEG2: MBZ.
6	31:8	Reserved (Reserved for encoder turbo mode **IntraResidueDataSize**, when this is not 0, optional residue data are provided to the PAK; Reserved for decoder)
	7:0	**MbIntraStruct** The IntraPredAvailFlags[4:0] have already included the effect of the constrained_intra_pred_flag. See the diagram later for the definition of neighbors position around the current MB or MB pair (in MBAFF mode). 1 – IntraPredAvailFlagX, indicates the values of samples of neighbor X can be used in intra prediction for the current MB. 0 – IntraPredAvailFlagX, indicates the values of samples of neighbor X is not available for intra prediction of the current MB. IntraPredAvailFlag-A and -E can only be different from each other when constrained_intra_pred_flag is equal to 1 and mb_field_decoding_flag is equal to 1 and the value of the mb_field_decoding_flag for the macroblock pair to the left of the current macroblock is equal to 0 (which can only occur when MbaffFrameFlag is equal to 1). IntraPredAvailFlag-F is used only if o it is in MBAFF mode, i.e. **MbaffFrameFlag** = 1 o the current macroblock is of frame type**,** i.e. **MbFieldFag** = 0, and o the current macroblock type is Intra8x8,

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>that is, IntraMbFlag = INTRA, IntraMbMode = INTRA_8x8, and Transform8x8Flag = 1.

In any other cases IntraPredAvailFlag-A shall be used instead.

<table>
<tr><th>Bits</th><th>IntraPredAvailFlags[4:0] Definition</th></tr>
<tr><td>7</td><td>IntraPredAvailFlagF – F (Left 8th row (-1,7) neighbor)</td></tr>
<tr><td>6</td><td>IntraPredAvailFlagA – A
(Left neighbor top half)</td></tr>
<tr><td>5</td><td>IntraPredAvailFlagE – E
(Left neighbor bottom half)</td></tr>
<tr><td>4</td><td>IntraPredAvailFlagB – B
(Top neighbor)</td></tr>
<tr><td>3</td><td>IntraPredAvailFlagC – C
(Top right neighbor)</td></tr>
<tr><td>2</td><td>IntraPredAvailFlagD – D
(Top left corner neighbor)</td></tr>
<tr><td>1:0</td><td>ChromaIntraPredMode –
2 bits to specify 1 of 4 chroma intra prediction mode, see the table in later section.</td></tr>
</table>
</td></tr>
</table>

Inline data subfields for an Inter Macroblock

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td rowspan="3">4</td><td>31:24</td><td>Reserved: MBZ (DXVA Decoder)</td></tr>
<tr><td>23:16</td><td>Reserved: MBZ (DXVA Decoder)</td></tr>
<tr><td>15:8</td><td>SubMbPredModes[bit 7:0] (Sub Macroblock Prediction Mode)

This field describes the prediction mode of the sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order.

This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA)

This field is derived from MbType for a non-BP_8x8 inter macroblock, and carries redundant information as MbType)

Bits [1:0]: SubMbPredMode[0] – for 8x8 Block 0

Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1

Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2

Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3

Blocks of the MB is numbered as follows :

0 1

2 3</td></tr>
</table>

DWord	Bit	Description
		Each 2-bit value [1:0] is defined as : 00 – Pred_L0 01 – Pred_L1 10 – BiPred **For VC1:** Bits [1:0]: "00"= Y0 Forward only, "01"= Y0 Backward only, "10"= Y0 Bi direction Bits [3:2]: SubMbPredMode[1] – for 8x8 Block 1 Bits [5:4]: SubMbPredMode[2] – for 8x8 Block 2 Bits [7:6]: SubMbPredMode[3] – for 8x8 Block 3
	7:0	**SubMbShape**[bit 7:0] (Sub Macroblock Shape) This field describes the sub-block partitioning of each sub macroblocks (four 8x8 blocks). It contains four subfields each with 2-bits, corresponding to the 4 fixed size 8x8 sub macroblocks in sequential order. This field is provided for MB with sub_mb_type equal to BP_8x8 only (B_8x8 and P_8x8 as defined in DXVA) This field is forced to 0 for a non-BP_8x8 inter macroblock, and effectively carries redundant information as MbType). Bits [1:0]: SubMbShape[0] – for 8x8 Block 0 Bits [3:2]: SubMbShape[1] – for 8x8 Block 1 Bits [5:4]: SubMbShape[2] – for 8x8 Block 2 Bits [7:6]: SubMbShape[3] – for 8x8 Block 3 Blocks of the MB is numbered as follows : 0 1 2 3 Each 2-bit value [1:0] is defined as : 00 – SubMbPartWidth=8, SubMbPartHeight=8 01 – SubMbPartWidth=8, SubMbPartHeight=4 10 – SubMbPartWidth=4, SubMbPartHeight=8 11 – SubMbPartWidth=4, SubMbPartHeight=4 For VC-1, This field indicates the transformation types used for luma components, 2 bits for each 8x8.
5	31:24	**Frame Store ID L0[3]**

DWord	Bit	Description
		Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	23:16	**Frame Store ID L0[2]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	15:8	**Frame Store ID L0[1]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation).
	7:0	**Frame Store ID L0[0]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index

DWord	Bit	Description
		are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
6	31:24	**Frame Store ID L1[3]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	23:16	**Frame Store ID L1[2]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	15:8	**Frame Store ID L1[1]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format.

DWord	Bit	Description
		Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)
	7:0	**Frame Store ID L1[0]** Support up to 4 Frame store ID per L0 direction, one per MB partition, if exists. See details in later section. This field specifies the frame Store ID into the Reference Picture List0 Table. Bit 7: **Must Be One**: (This is reserved for control fields in future extension, when reference index are generated instead of frame store ID) 1: indicate it is in Frame store ID format. 0: indicate it is in Reference Index format. Bit 6:5: reserved MBZ Bit 4:0: Frame store index or Frame Store ID (Bit 4:1 is used to form the binding table index in intel implementation)

AVC Encoder PAK Commands

Each PAK Commands is composed of a command op-code DW and one or more command data DWs (inline data). The size of each command is specified as part of the op-code DW. Most of the commands have fixed size, except some are allowed to be of variable length.

There is an inherent order of executing MFC PAK commands that driver must follow.

MFC_AVC_PAK_OBJECT

Indirect Data Description

For each macroblock, an ENC-PAK data set consists of two types of data blocks: indirect **MV data block** and **inline MB information**.

The indirect MV data block may be in two modes: **unpackedmode** and **packed-size mode**.

Unpacked Motion Vector Data Block

In the **unpacked** mode, motion vectors are expanded (or duplicated) to either bidirectional 8x8 8MV major partition format, or bidirectional 4x4 32MV format. Thus either 32 bytes or 128 bytes is assigned to each MB.

Motion Vector block contains motion vectors in an intermediate format that is partially expanded according to the sub- macroblock size. During the expansion, a place that does not contain a motion vector is filled by replicating the relevant motion vector according to the following motion vector replication rules. If the relevant motion vector doesn't exist (for the given L0 or L1), it is zero filled.

Motion Vector Replication Rules:

- Rule #1
    - #1.1: For L0 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L0 inter prediction exists, the corresponding L0 MV is used
        - Else it must be zero
    - #1.2: For L1 MV, for any sub-macroblock or sub-partition where there is at least one motion vector
        - If L1 inter prediction exists, the corresponding L1 MV is used
        - Else it must be zero
- For a macroblock with a 16x16, 16x8 or 8x16 sub-macroblock, MvSize = 8. The eight MV fields follow Rule #1.
    - The 16x16 is broken down into 4 8x8 sub-macroblocks. The 16x16 MVs (after rule #1) are replicated into all 8x8 blocks.
    - For an 8x16 partition, each 8x16 is broken down into 2 8x8 stacking vertically. The 8x16 MVs (after rule #1) are replicated into both 8x8 blocks.
    - For a 16x8 partition, each 16x8 is broken down into 2 8x8 stacking horizontally. The 16x8 MVs (after rule #1) are replicated into both 8x8 blocks.
- For macroblock with sub-macroblock of 8x8 without minor partition (SubMbShape[0…3] = 0), MvSize = 8, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
    - There is no motion vector replication
- For macroblock with sub-macroblock of 8x8 with at least one minor partition (if any SubMbShape[i] != 0), MvSize = 32, (e.g. mb_type equal to P_8x8, P_8x8ref0, or B_8x8)
    - For an 8x8 sub-partition, the 8x8 MVs (after rule #1) is replicated into all the four 4x4 blocks.
    - For an 4x8 sub-partition within an 8x8 partition, each 4x8 is broken down into 2 4x4 stacking vertically. The 4x8 MVs (after rule #1) are replicated into both 4x4 blocks.
    - For an 8x4 sub-partition within an 8x8 partition, each 8x4 is broken down into 2 4x4 stacking horizontally. The 8x4 MVs (after rule #1) are replicated into both 4x4 blocks.
    - For a 4x4 sub-partition within an 8x8 partition, each 4x4 has its own MVs (after rule #1).

Motion Vector block and MvSize

DWord	Bit	MvSize	
		8	32
W1.0	31:16	**MV_Y0_L0.y**	**MV_Y0_0_L0.y**
	15:0	**MV_Y0_L0.x**	**MV_Y0_0_L0.x**
W1.1	31:16	**MV_Y0_L1.y**	**MV_Y0_0_L1.y**
	15:0	**MV_Y0_L1.x**	**MV_Y0_0_L1.x**
W1.2	31:0	**MV_Y1_L0**	**MV_Y0_1_L0**
W1.3	31:0	**MV_Y1_L1**	**MV_Y0_1_L1**
W1.4	31:0	**MV_Y2_L0**	**MV_Y0_2_L1**
W1.5	31:0	**MV_Y2_L1**	**MV_Y0_2_L0**
W1.6	31:0	**MV_Y3_L0**	**MV_Y0_3_L0**
W1.7	31:0	**MV_Y3_L1**	**MV_Y0_3_L1**
W2.0	31:0	**n/a**	**MV_Y1_0_L1**
W2.1	31:0	**n/a**	**MV_Y1_0_L0**
W2.2	31:0	**n/a**	**MV_Y1_1_L1**
W2.3	31:0	**n/a**	**MV_Y1_1_L0**
W2.4	31:0	**n/a**	**MV_Y1_2_L1**
W2.5	31:0	**n/a**	**MV_Y1_2_L0**
W2.6	31:0	**n/a**	**MV_Y1_3_L0**
W2.7	31:0	**n/a**	**MV_Y1_3_L1**
W3.0	31:0	**n/a**	**MV_Y2_0_L1**
W3.1	31:0	**n/a**	**MV_Y2_0_L0**

DWord	Bit	MvSize	
		8	32
W3.2	31:0	**n/a**	**MV_Y2_1_L1**
W3.3	31:0	**n/a**	**MV_Y2_1_L0**
W3.4	31:0	**n/a**	**MV_Y2_2_L1**
W3.5	31:0	**n/a**	**MV_Y2_2_L0**
W3.6	31:0	**n/a**	**MV_Y2_3_L0**
W3.7	31:0	**n/a**	**MV_Y2_3_L1**
W4.0	31:0	**n/a**	**MV_Y3_0_L1**
W4.1	31:0	**n/a**	**MV_Y3_0_L0**
W4.2	31:0	**n/a**	**MV_Y3_1_L1**
W4.3	31:0	**n/a**	**MV_Y3_1_L0**
W4.4	31:0	**n/a**	**MV_Y3_2_L1**
W4.5	31:0	**n/a**	**MV_Y3_2_L0**
W4.6	31:0	**n/a**	**MV_Y3_3_L0**
W4.7	31:0	**n/a**	**MV_Y3_3_L1**

The motion vector(s) for a given sub-macroblock or a sub-partition are uniquely placed in the output message as shown by the non-duplicate fields in *Unpacked Motion Vector Data Block* and *Unpacked Motion Vector Data Block*.

MV_Yx_L0 and MV_Yx_L1 may be present individually or both. If one is not present, the corresponding field must be zero. Subsequently, the duplicated fields will be zero as well.

Motion Vector duplication by sub-macroblocks for a 16x16 macroblock, whereas the 8x8 column is for 4x(8x8) partition without minor shape

DWord	Bit	16x16	16x8	8x16	8x8
W1.0	31:16	**MV_Y0_L1 (A)**	**MV_Y0_L1 (A)**	**MV_Y0_L1**	**MV_Y0_L1**
	15:0	**MV_Y0_L0 (A)**	**MV_Y0_L0 (A)**	**MV_Y0_L0**	**MV_Y0_L0**
W1.1	31:16	**Duplicate (A)**	**Duplicate (A)**	**MV_Y1_L1**	**MV_Y1_L1**
	15:0	**Duplicate (A)**	**Duplicate (A)**	**MV_Y1_L0**	**MV_Y1_L0**
W1.2	31:16	**Duplicate (A)**	**MV_Y2_L1 (B)**	**Duplicate (A)**	**MV_Y2_L1**
	15:0	**Duplicate (A)**	**MV_Y2_L0 (B)**	**Duplicate (A)**	**MV_Y2_L0**
W1.3	31:16	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y3_L1**
	15:0	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y3_L0**

Motion Vector duplication by sub-partitions for the first 8x8 sub-macroblock Y0 if any Y0-Y3 contains minor shape (Y1_ to Y3_ have the same format in W2 to W4)

DWord	Bit	8x8	8x4	4x8	4x4
W1.0	31:16	**MV_Y0_L1**	**MV_Y0_0_L1 (A)**	**MV_Y0_0_L1 (A)**	**MV_Y0_0_L1**
	15:0	**MV_Y0_L0**	**MV_Y0_0_L0 (A)**	**MV_Y0_0_L0 (A)**	**MV_Y0_0_L0**
W1.1	31:16	**Duplicate (A)**	**Duplicate (A)**	**MV_Y0_1_L1 (B)**	**MV_Y0_1_L1**
	15:0	**Duplicate (A)**	**Duplicate (A)**	**MV_Y0_1_L0 (B)**	**MV_Y0_1_L0**
W1.2	31:16	**Duplicate (A)**	**MV_Y0_2_L1 (B)**	**Duplicate (A)**	**MV_Y0_2_L1**
	15:0	**Duplicate (A)**	**MV_Y0_2_L0 (B)**	**Duplicate (A)**	**MV_Y0_2_L0**
W1.3	31:16	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y0_3_L0**
	15:0	**Duplicate (A)**	**Duplicate (B)**	**Duplicate (B)**	**MV_Y0_3_L1**

Packed-Size Motion Vector Data Block

In the packed case, no redundant motion vectors are sent. So the number of motion vectors sent, as specified by **MvQuantity** is the same as the motion vectors that will be packed (**MvPacked**).

The following tables are for information only. Fields like MvQuantity and MvPacked are not required interface fields.

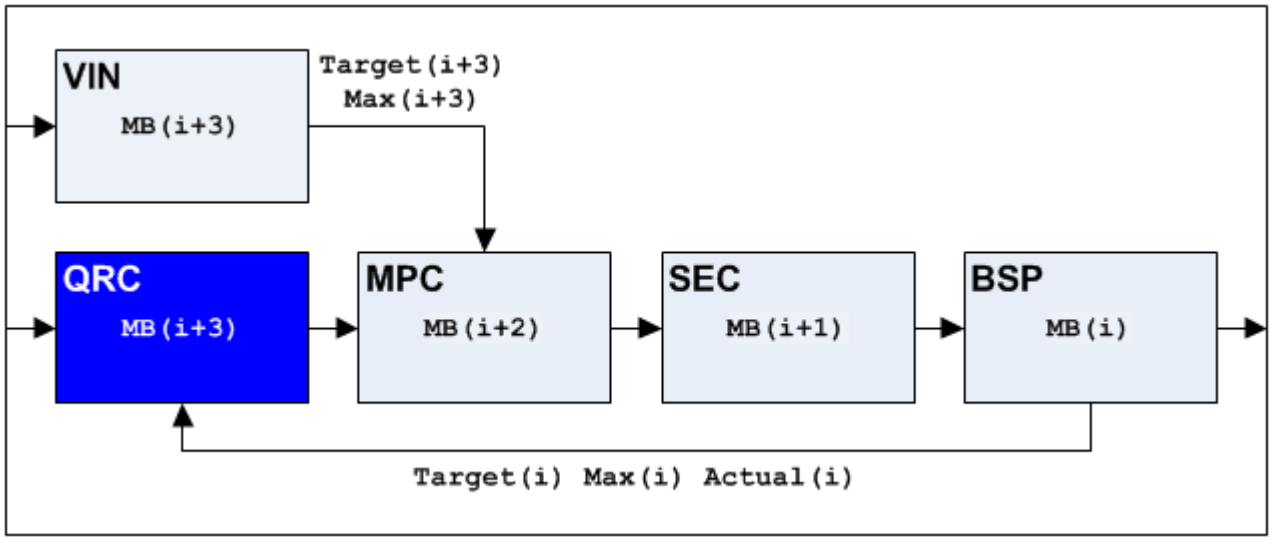
MbSkipFlag	MbType	Description	Mv Quantity	MvSize	(Minimal MvSize)
1	1	P_Skip_16x16	0	8	1
0	1	BP_L0_16x16	1	8	1
0	2	B_L1_16x16	1	8	1
0	3	B_Bi_16x16	2	8	2
0	4	BP_L0_L0_16x8	2	8	4
0	5	BP_L0_L0_8x16	2	8	4
0	6	B_L1_L1_16x8	2	8	8
0	7	B_L1_L1_8x16	2	8	8
0	8	B_L0_L1_16x8	2	8	8
0	9	B_L0_L1_8x16	2	8	8
0	0Ah	B_L1_L0_16x8	2	8	8
0	0Bh	B_L1_L0_8x16	2	8	8
0	0Ch	B_L0_Bi_16x8	3	8	8
0	0Dh	B_L0_Bi_8x16	3	8	8
0	0Eh	B_L1_Bi_16x8	3	8	8
0	0Fh	B_L1_Bi_8x16	3	8	8
0	10h	B_Bi_L0_16x8	3	8	8
0	11h	B_Bi_L0_8x16	3	8	8
0	12h	B_Bi_L1_16x8	3	8	8
0	13h	B_Bi_L1_8x16	3	8	8
0	14h	B_Bi_Bi_16x8	4	8	8
0	15h	B_Bi_Bi_8x16	4	8	8
0	16h	**BP**_8x8	[3]4	8 or 32	8 or 32

When MbType = 22, BP_8x8, take the sum of four individual 8x8 subblocks

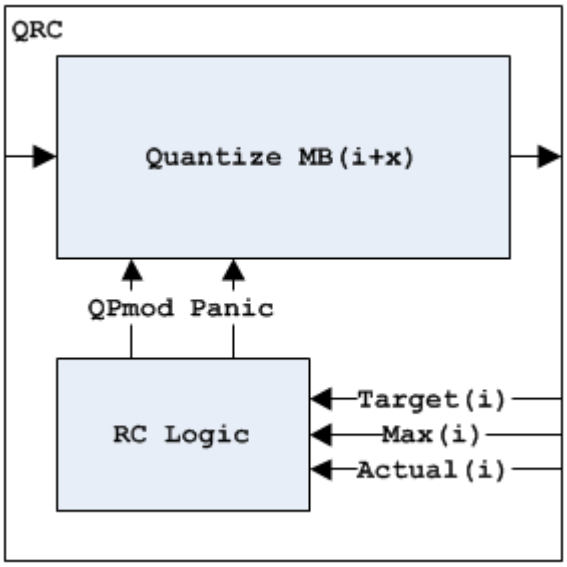
Direct8x8Pattern	SubMb Shape	SubMb PredMode	Description	Mv Quantity	Mv Size	(Min MvSize)
OR	OR	OR		ADD	ADD	ADD
1	0	0	P_Skip_8x8 B_Direct_L0_8x8 (B-Skip_ L0_8x8)	0	2	1
1	0	1	B_Direct_L1_8x8 (B-Skip_ L1_8x8)	0	2	1
1	0	2	B_Direct_Bi_8x8 (B-Skip_ Bi_8x8)	0	2	2
1	3	0	P_Skip_4x4 B_Direct_L0_4x4 (B-Skip_ L0_4x4)	0	8	4
1	3	1	B_Direct_L1_4x4 (B-Skip_ L1_4x4)	0	8	4
1	3	2	B_Direct_Bi_4x4 (B-Skip_ Bi_4x4)	0	8	8
0	0	0	BP_L0_8x8	1	2	1
0	0	1	B_L1_8x8	1	2	1
0	0	2	B_BI_8x8	2	2	2
0	1	0	BP_L0_8x4	2	8	4
0	1	1	B_L1_8x4	2	8	4
0	1	2	B_BI_8x4	4	8	8
0	2	0	BP_L0_4x8	2	8	4
0	2	1	B_L1_4x8	2	8	4
0	2	2	B_BI_4x8	4	8	8
0	3	0	BP_L0_4x4	4	8	4
0	3	1	B_L1_4x4	4	8	4
0	3	2	B_BI_4x4	8	8	8

Macroblock Level Rate Control

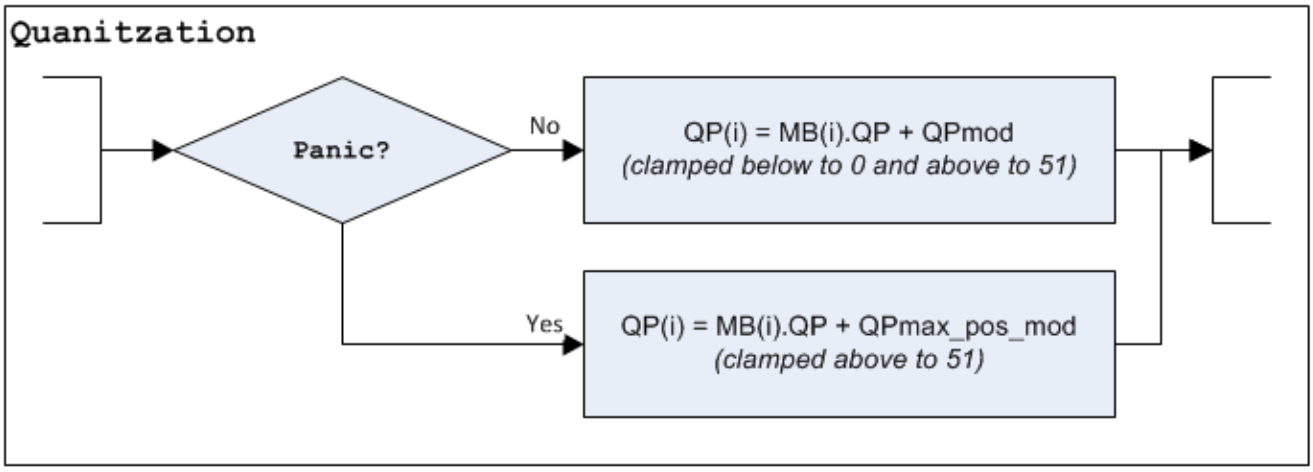
The QRC (Qauntization Rate Control) unit receives data from BSP (Bit Serial Packer) and VIN (Video In) and generates adjustments to QP values across macroblocks.



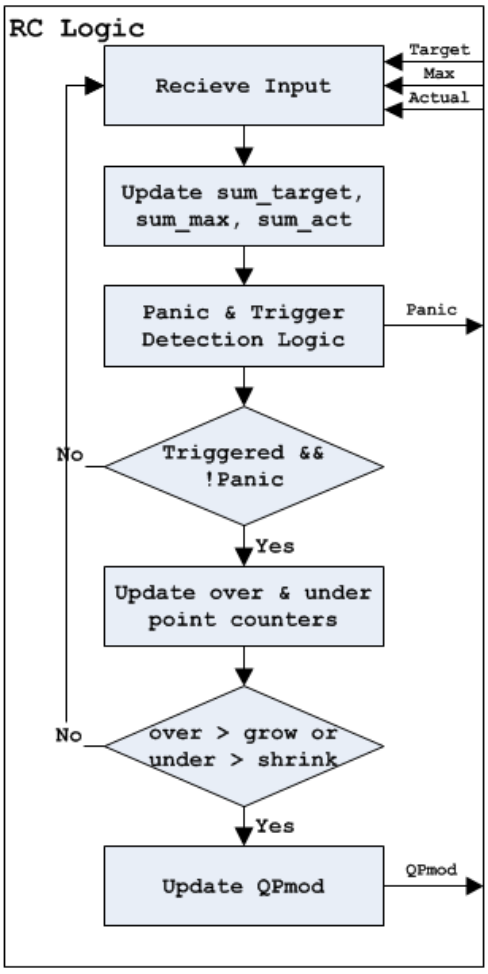
QRC can be logically partitioned into two units as shown below.



Macroblock level rate control is handled by the RC logic and the quantization logic.



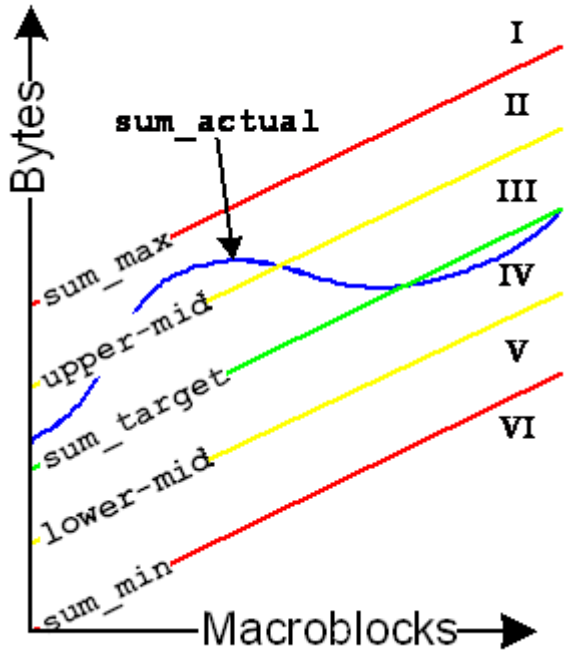
The signals QPmod and panic are generated by the RC logic based on data feedback from BSP. A flowchart of the RC logic is given below.



Theory of Operation Overview

BSP will generate a byte estimate for each macroblock packed. Additionally, the user will specify a target and max size per macroblock. The running sum of these signals (actual, target, max) creates "curves" which are used to identify when QP adjustments are necessary (see figure below). Three more curves are symmetrically generated by QRC (upper_midpt, lower_midpt, sum_min) from target and max. The values of target and max are specified by the user will dictate the shape of these curves.

The difference between sum_actual and sum_target (called 'bytediff') identifies the margin of error between the target and actual sizes. The difference between the current bytediff and the previously calculated bytediff represents the rate of change in this margin over time. The sign of this rate is used to identify if the correction is trending in the appropriate direction (towards bytediff = 0).



QPmod

Each macroblock will have a requested QP (which could vary across macroblocks or remain constant). QPmod is to be added to the QP requested. QPmod will be positive when the target was under-predicted and negative when the target is over-predicted.

QPmod is incremented or decremented when internal counters (called 'over' and 'under') reach tripping points (called 'grow' and 'shrink'). For each MB processed and based on which region (1-6) sum_actual falls in, various amounts of points are added to either counters. If over exceeds grow, QPmod is incremented whereas if under exceeds shrink, QPmod is decremented.

To dampen the effect of repeated changes in the same direction, an increase in resistance for that direction and decrease in resistance for the complementary direction occurs (called 'grow_resistance' and 'shrink_resistance'). This resistance is added to grow or shrink, which then requires more points to trip the next correction in that direction.

The user can specify guard-bands that limit the amount QPmod can be modified. QPmod cannot exceed QPmax_pos_mod or become less than -QPmax_neg_mod_abs.

Triggering

The RC unit begins to modify QPmod occurs only when it is triggered.

Three levels of triggering exist: always, gentle, loose. Always means that RC will be active once sum_actual reaches regions 3 or 4. Gentle will trigger RC once sum_actual reaches regions 2 or 5. Loose waits to trigger RC when sum_actual reaches regions 1 or 6.

RC will deactivate (triggered = false) once sum_actual begins to track sum_target over a series of macroblocks. Specifically, the sign of the rate of change for bytediff is monitored over a window of macroblocks. When the sum of these signs over the window falls within a tolerance value (called 'stable'), triggered will reset to false.

Panic

When enabled, panic mode will occur whenever sum_actual reaches region 1 and will remain so until sum_actual reaches region 4. When panicking, all macroblocks will be quantized with QP = MB(n).QP + QPmax_pos_mod, clamped to 51.

User Controls

This unit achieves a large flexibility by allowing the user to define various key parameters. At the per-macroblock level, the values of target and max are specifed and will create various shapes of curves that sum_actual will be compared against.

Per-slice, the user can specify the triggering sensitivity and the tolerance required to disable the trigger. Additionally, the user can enable panic detection.

The point values assigned to each of the 6 regions are exposed to the user which allow for asymmetrical control for over and under predictions amongst other things. Additionally, the user can specify the initial values of grow and shrink along with the resistance values applied when correction is invoked.

Lastly, the maximum and minimum values for QPmod are also exposed to the user.

AVC Encoder MBAFF Support

1. **Algorithm**

   Prediction of current macroblock motion vector is possible from neighboring macroblocks mbAddrA/mbAddrD/mbAddrB/mbAddrC/mbAddrA+1/mbAddrD+1/mbAddrB+1/mbAddrC+1. The selection of these macroblocks depends on coding type(field/frame) of current macroblock pair and the coding of neighboring macroblock pair.

   Selection of these macroblock pairs is described in detail in following sections.

   1.1 Selection of Top LeftMB pair: The selection of Top Left MB pair depends on coding type of current and also top left macroblock pair.

1.2 Selection of LeftMB pair: The selection of Left MB pair depends on coding type of current and also left macroblock pair.

1.3 Selection of Top MB pair: The selection of Top MB pair depends on coding type of current and also top macroblock pair.

1.4 Selection of Top RightMB pair: The selection of Top Right MB pair depends on coding type of current and also top right macroblock pair.

1.5 Motion Vector and refIdx Scaling: Motion vectors and reference index of neighboring macroblocks (mbAddrA/mbAddrB/mbAddrC/mbAddrD) should be scaled before using them into prediction equations. Again the scaling depends on coding type of current and neighboring macroblock pair which is described as follows

- If the current macroblock is a field macroblock and the macroblock mbAddrN is a frame macroblock ...

```
mvLXN[ 1 ] = mvLXN[ 1 ] / 2 (8-214)
refIdxLXN = refIdxLXN * 2   (8-215)
```

- Otherwise, if the current macroblock is a frame macroblock and the macroblock mbAddrN is a field macroblock ...

```
mvLXN[ 1 ] = mvLXN[ 1 ] * 2 (8-216)
refIdxLXN = refIdxLXN / 2   (8-217)
```

- Otherwise, the vertical motion vector component mvLXN[1] and the reference index refIdxLXN remain unchanged.

MPEG-2

MPEG2 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_MPEG2_PIC_STATE

MPEG2 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_MPEG2_BSD_OBJECT

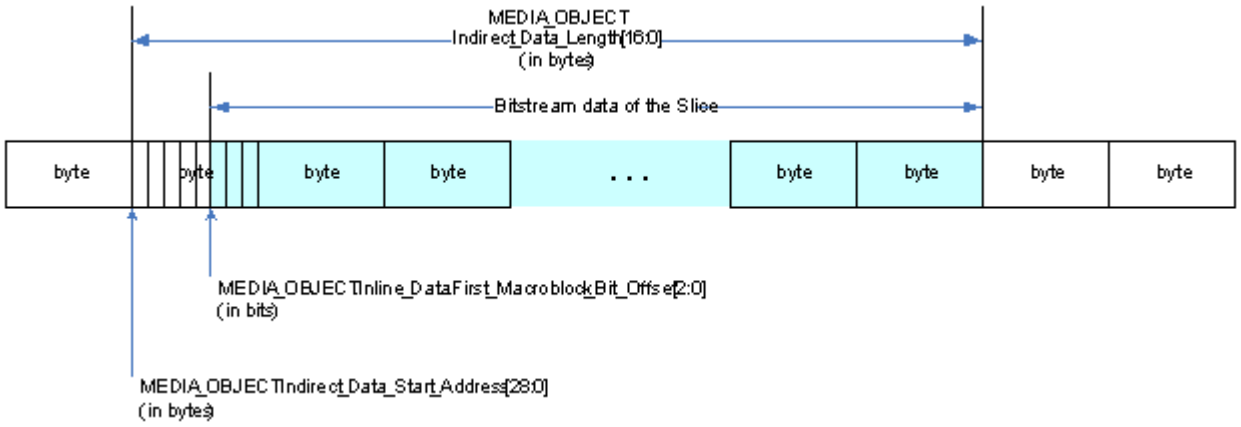
MFD_MPEG2_BSD_OBJECT Inline Data Description

Indirect Data Description

The indirect data start address in MFD_MPEG2_BSD_OBJECT specifies the starting Graphics Memory address of the bitstream data that follows the slice header. It provides the byte address for the first macroblock of the slice. Together with the First Macroblock Bit Offset field in the inline data, it provides the bit location of the macroblock within the compressed bitstream.

The indirect data length in MFD_MPEG2_BSD_OBJECT provides the length in bytes of the bitstream data for this slice. It includes the first byte of the first macroblock and the last **non-zero** byte of the last macroblock in the slice. Specifically, the zero-padding bytes (if present) and the next start-code are excluded. Hardware ignores the contents after the last non-zero byte. *Indirect Data Description* illustrates these parameters for a slice data.

Indirect data buffer for a slice



MPEG2 Encoder PAK Commands

The MFC_MPEG2_PAK_INSERT_OBJECT Command is identical to the MFC_AVC_PAK_INSERT_OBJECT command as described in this document.

The MFC_MPEG2_STITCH_OBJECT Command is identical as MFC_AVC_STITCH_OBJECT command as described in this document.

MFC_MPEG2_SLICEGROUP_STATE

MFC_MPEG2_PAK_OBJECT

PAK Object Inline Data Description – MPEG-2

The Inline Data includes all the required MB encoding states, constitute part of the Slice Data syntax elements, MB Header syntax elements and their derivatives. It provides information for the following operations:

1. Forward and Inverse Transform
2. Forward and Inverse Quantization
3. Advanced Rate Control (QRC)
4. MB Parameter Construction (MPC)
5. VLC encoding
6. Bit stream packing
7. Internal error handling

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFC_MPEG2_PAK_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_MPEG2_PAK_OBJECT command.

The inline data has been designed to match AVC MB structure for efficient transcoding.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

DWord	Bit	Description
1	31:27	Reserved: MBZ
	22-20	**MvFormat (Motion Vector Size)**. This field specifies the size and format of the input motion vectors. This field is reserved (MBZ) when the **IntraMbFlag** = 1. The valid encodings are: **011 = Unpacked: Two motion vector pairs** Others are reserved. *(The following encodings are intended for other formats:* *001 = 1MV: one 16x16 motion vector* *010 = 2MV: One 16x16 motion vector pair* *011 = 4MV: Four 8x8 motion vectors, or Two 16x8 motion vector pairs* *100 = 8MV: Four 8x8 motion vector pairs* *101 = 16MV: 16 4x4 motion vectors* *110 = 32MV: 16 4x4 motion vector pairs* *111 = Packed, number of MVs is given by **packedMvNum**.)*
	19	**CbpDcY.** This field specifies if the Luma DC coded. Must be 1 for MPEG-2.
	18	**CbpDcU.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.**
	17	**CbpDcV.** This field specifies if the Chroma Cb DC coded. Must be 1 for MPEG-2**.**
	16	Reserved: MBZ
	15	**TransformFlag** Used to indicate transformation type for MPEG-2. 0 = Frame DCT transformation 1 = Field DCT transformation
	14	**FieldMbFlag** For MPEG-2, this flag is set to 1 if either the picture is in field type or the MB is INTER of field type, i.e. split into two 16x8 field blocks.

	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. For I-picture MB (IntraPicFlag =1), this field must be set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes). 0: INTER (inter macroblock) 1: INTRA (intra macroblock)
	12:8	**MbType** This field is encoded to match with the best macroblock mode determined as described in the next section. It follows an unified encoding for inter and intra macroblocks according to MFX Encoding reference as shown in Figure A.
	7:3	Reserved : MBZ
	2	**SkipMbFlag** By setting it to 1, this field forces an inter macroblock to be encoded as a skipped macroblock. It is equivalent to mb_skip_flag in AVS spec, Hardware honors input MVs for motion prediction and forces CBP to zero. By setting it to 0, an inter macroblock will be coded as a normal inter macroblock. The macroblock may still be coded as a skipped macroblock, according to the macroblock type conversion rules described in the later sub sections. This field can only be set to 1 for certain values of MbType. See details later. This field is only valid for an inter macroblock. Hardware ignores this field for an intra macroblock. 0 = not a skipped macroblock 1 = is coded as a skipped macroblock **Note:** When this flag is set to 1, the correct MVs are assumed for HW decoder to generate decoded reconstruction frame.
	1:0	**InterMbMode** This field is provided to carry redundant information as that encoded in MbType. This field is only valid if **IntraMbFlag** =0, otherwise, it is ignored by hardware.
2	31:16	**MbYCnt (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. Format = U16 in unit of macroblock.
	15:0	**MbXCnt (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U16 in unit of macroblock.

3	31:24	**MaxSizeInWord** PAK should not exceed this budget accumulatively, otherwise it will trickle the PANIC mode.
	23:16	**TargetSizeInWord** PAK should use this budget accumulatively to decide if it needs to limit the number of non-zero coefficients.
	15:13	**MBZ**
	12:0	**Cbp –** Coded Block Pattern. This field specifies whether blocks are present or not. Format = 6-bit mask (or 8-bit, & 12-bit, for 422 and 444). Bit 11: Y0Bit 10: Y1Bit 9: Y2Bit 8: Y3 Bit 7: Cb4Bit 6: Cr5Bits 0-5: MBZ
4	31	**LastMbInSlice –** the last MB in a slice.
	30	**FirstMbInSlice –** the first slice in a slice, it requires slice header insertion.
	29:28	**MBZ**
	27	**EnableCoeffClamp** 1 = the magnitude of coefficients of the current MB will be clamped based on the clamping matrix after quantization 0 = no clamping
	26	**LastMbInSG** 1 – the current MB is the last MB in the current slice group.
	25	**MbSkipConvDisable** This is a per-MB level control to enable and disable skip conversion. This field is ORed with SkipConvDisable field. This field is only valid for a P or B slice. It must be zero for other slice types. Rules are provided in Macroblock Type Conversion Rules 0 - Enable skip type conversion for the current macroblock 1 – Disable skip type conversion for the current macroblock
	24	**FirstMbInSG** 1 – the current MB is the last MB in the current slice group.
	23:20	**MBZ**

<table>
<tr><td rowspan="3"></td><td>19:16</td><td>MvFieldSelect – Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the ISO/IEC 13818-2 (see also §7.6.4).
<table><tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr><tr><td>16</td><td>0</td><td>0</td><td>0</td></tr><tr><td>17</td><td>0</td><td>1</td><td>1</td></tr><tr><td>18</td><td>1</td><td>0</td><td>2</td></tr><tr><td>19</td><td>1</td><td>1</td><td>3</td></tr></table>Format = MC_MotionVerticalFieldSelect.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td>15:5</td><td>MBZ Reserved</td></tr>
<tr><td>4:0</td><td>QpScaleCode</td></tr>
<tr><td>5</td><td>31:16</td><td>MV[0][0].y – the y coordinate of the first forward MV
if Mv[0][0] n/a:
if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].
else it MUST be set to the value 0</td></tr>
<tr><td></td><td>15:0</td><td>MV[0][0].x – the x coordinate of the first forward MV
if Mv[0][0] n/a:
if Mv[1][0] available, it MUST be set to the same value as Mv[1][0].
else it MUST be set to the value 0</td></tr>
<tr><td>6</td><td>31:0</td><td>MV[1][0] – the first backward MV
if Mv[1][0] n/a: it MUST be set to the same value as Mv[0][0]</td></tr>
<tr><td>7</td><td>31:0</td><td>MV[0][1] – the second forward MV
if Mv[0][1] n/a:
if Mv[1][1] available, it MUST be set to the same value as Mv[1][1].
else it MUST be set to the same value as Mv[0][0]</td></tr>
<tr><td>8</td><td>31:0</td><td>MV[1][1] – the second backward MV
if Mv[1][1] n/a: it MUST be set to the same value as Mv[1][0]</td></tr>
</table>

The mapping between MPEG-2 spec and MfxMbCode can be achieved according to the following:

1) Renamed variables with identical meaning:

MPEG-2 Spec	MFX API	Value
macroblock_quant	**MbQuantPresent**	0 or 1
macroblock_intra	**IntraMbFlag**	0 or 1
dct_type	**Transform8x8Flag**	0 or 1
macroblock_pattern	**Cbp8x8**	remapped

2) Macroblock type remapping:

		B-spec Entry					MPEG-2 Spec				
Frame Type	**Mb Type**	**Intra Mb Flag**	**Skip Mb Flag**	**Mb Type 5Bits**	**Field Mb Flag**	**Inter Mb Mode**	**macroblock_intra**	**motion_type_bit 0**	**motion_type_bit 1**	**motion_forward**	**motion_backward**
IPB	Intra	1	0	1Ah	0/1	-	1	-	-	-	-
P B B	Skip	0	1	01h 02h 03h	0/1	0	0	-	-	1 0 1	0 1 1
P	0-MV*	0	0	01h	0/1	0	0	-	-	0	0
P Frame	Frame type	0	0	01h	0	0	0	0	1	1	0
P Frame	Field type	0	0	04h	1	1	0	1	0	1	0
P Frame	dual prime	0	0	19h	0	0	0	1	1	1	0
P Field	One 16x16	0	0	01h	1	0	0	1	0	1	0
P Field	Two 16x8	0	0	04h	1	1	0	0	1	1	0
P Field	dual prime	0	0	19h	1	0	0	1	1	1	0

		B-spec Entry					MPEG-2 Spec				
Frame Type	Mb Type	Intra Mb Flag	Skip Mb Flag	Mb Type 5Bits	Field Mb Flag	Inter Mb Mode	macroblock_intra	motion_type_bit0	motion_type_bit1	motion_forward	motion_backward
B Frame	Frame type	0	0	01h 02h 03h	0	0	0	0	1	1 0 1	0 1 1
B Frame	Field type	0	0	04h 06h 14h	1	1	0	1	0	1 0 1	0 1 1
B Field	One 16x16	0	0	01h 02h 03h	1	0	0	1	0	1 0 1	0 1 1
B Field	Two 16x8	0	0	04h 06h 14h	1	1	0	0	1	1 0 1	0 1 1

- Notice that there is no special way to indicate 0 motion vector case for P frame. It is for PAK to handle internally by checking up the motion vector values.
- Notice also, the MbType5bits is adapted from AVC DXVA macroblock types. It may seems awkward from MPEG-2 perspective, but provides a common VME interface for us for simpler HW design and help the advanced transcoding solution.

MFX HW Interface and DXVA Conversion

This topic is currently under development.

Map DXVA to HW BSpec

		HW	
Location		BSPEC	
BYTE		MPEG-2	DXVA
	DW0		
0		**MbMode**	
0.0-1	0[0-1]	**InterMbMode**	see (A)
0.2	0[2]	**SkipMbFlag**	<-MBskipsFollowing
0.3	0[3]	**mbz**	
0.4-5	0[4-5]	**IntraMbMode**	IntraMacroblock
0.6	0[6]	**mbz**	
0.7	0[7]	**FieldMbPolarity**	derived
1		**MbType**	
1.0-4	0[8-12]	**MbType5Bits**	see (A)
1.5	0[13]	**IntraMbFlag**	IntraMacroblock
1.6	0[14]	**FieldMbFlag**	see (A)
1.7	0[15]	**TransformFlag**	FieldResidual
2		**MbFlag**	
2.0	0[16]	**ResidDataFlag**	HostResDiff
2.1	0[17]	**CbpDcV**	PAK control
2.2	0[18]	**CbpDcU**	PAK control
2.3	0[19]	**CbpDcY**	PAK control
2.4-6	0[20-22]	**MvFormat**	= 3, derived
2.7	0[23]	**mbz**	
3	0[24-31]	**PackedMvNum**	see (A)
	DW1		
4-5	1[0-15]	**MbXCnt**	wMBaddress
6-7	1[16-31]	**MbYCnt**	wMBaddress
	DW2		
8	2[0-7]		bNumCoef[0]
8.0-5	2[0-5]	**mbz**	
8.6-7	2[6-7]	**CbpAcUV**	PAK control
9	2[8-11]	**CbpAcY**	PAK control

		HW	
Location		BSPEC	
BYTE		MPEG-2	DXVA
	2[12-15]	**mbz**	
10	2[16-23]	**TargetedSzInWord**	
11	2[24-31]	**MaxSzInWord**	
	DW3		
12		**Qscale**	derived
12.0-6	3[0-6]	**QScaleCode**	
12.7	3[7]	**QScaleType**	
13	3[8-15]	**mbz**	
14	3[16-19]	**MvFieldSelect**	`MvertFieldSel`
	3[20-23]	**mbz**	
15		**MbExtFlag**	
15.0	3[24]	**mbz**	
15.1	3[25]	**SkipMvConvDisable**	
15.2	3[26]	**LastMbFlag**	PAK control
15.3	3[27]	**EnableCoeffClamp**	PAK control
15.4-5	3[28-29]	**MbScanMethod**	`MBscanMethod`
15.6	3[30]	**NewSliceFlag**	PAK control
15.7	3[31]	**EndSliceFlag**	PAK control
	DW4-7		
16-32	4-7[all]	**MV[2][2][2]**	`MVector[4][2]`

(A): Set **InterMbMode, MbType5bits, FieldMbFlag**, and **PackedMvNum** from DXVA parameters:

```
 if(IntraMacroblock) return (TYPE_INTRA);
 else if(MotionType==3){ // dual prime
    MbType5bits = 0x19; FieldMbFlag = 0; InterMbMode = 0; PackedMvNum = 2;     return
(DUAL_PRIME);
 }
 else{
    IsFieldFrame = a PicState derivative;     switch(MotionType+IsFieldFrame{
       case 1: // Two 16x8 field in Frame Frame
       case 3: // Two 16x8 field in Field Frame
          FieldMbFlag = 1; InterMbMode= 1;          switch(MotionForward |Motionbackward
«1)){
             case 1:
                MbType5bits = 4; PackedMvNum = 2;                break;
             case 2:
                MbType5bits = 6; PackedMvNum = 2;                break;
             case 3:
                MbType5bits = 0x14; PackedMvNum = 4;                break;
          }
          break;
       case 2: // 16x16 block in either case
          FieldMbFlag = IsFieldFrame; InterMbMode = 0;
switch(MotionForward|(Motionbackward«1)){
             case 1:
                MbType5bits = 1; PackedMvNum = 1;                break;
             case 2:
                MbType5bits = 2; PackedMvNum = 1;                break;
             case 3:
                MbType5bits = 3; PackedMvNum = 2;                break;
          }
          break;
    }
 }
```

Map HW Bspec to DXVA

	DXVA	BSPEC
		MPEG-2
0-1	wMBaddress	**= MbYCnt*MbW + MbXCnt**
2-3	wMBtype	
2.0	IntraMacroblock	**= IntraMbFlag**
2.1	MotionForward	see (B)
2.2	MotionBackward	see (B)
2.3	Motion4MV	VC-1 only, MBZ for Mpeg-2
2.4	Reserved	
2.5	FieldResidual	**= TranformFlag**
2.6-7	MBscanMethod	**= MbScanMethod**
3.0-1	MotionType	see (B)
3.2	HostResDiff	**= ResidDataFlag**
3.3	Reserved	
3.4-7	MvertFieldSel	**= MvFieldSelect**
4	MBskipsFollowing	count **SkipMbFlag**
5-7	MBdataLocation	n/a
8-9	wPatternCode	**= CbpAcY|UV**
10-15	bNumCoef[6]	**n/a**
16-32	MVector[4][2]	**= MV[2][2][2]**

(B): Set **MBtype** and **MotionType** from Bspec interface

```
if(MbIntraFlag) return (TYPE_INTRA);
else if(MbType5Bits&8){ // dual prime
        MotionForward= 1;
        MotionBackward= 0;
        MotionType= 3;
              return (DUAL_PRIME);
        }
         else{
        // redundant: InterMbMode = !!(MbType5Bits&4);
        if(InterMbMode){
                    MotionForward= !(MbType5Bits&2);
                    MotionBackward= !!(MbType5Bits&0x12);
              }
               else{
                    MotionForward= (MbType5Bits&1);
                    MotionBackward= !!(MbType5Bits&2);
              }
              MotionType = 2-(InterMbMode^FieldMbFlag);
              // equivalently the 2 bits are:
              // MotionType0 = (InterMbMode^FieldMbFlag);
              // MotionType1 = ~MotionType0;
               return (TYPE_INTER);
}
```

VC1 Common Commands

MFX Commands are organized into groups based on their scope of functioning. There are Pipeline Common state commands that are common to all codecs (encoder and decoder) and is applicable to the processing of one full frame/field. There are also individual codec Common state commands that are common to both encoder and decoder of that particular codec. These latter common state commands, some are applicable at the processing of one full frame/field, and some are applicable at the processing of an individual slice level.

MFX_VC1_PRED_PIPE_STATE

MFX_VC1_DIRECTMODE_STATE

VC1 Decoder Commands

These are decoder-only commands. They provide the pointer to the compressed input bitstream to be decoded.

MFD_VC1_LONG_PIC_STATE

AltPQuantConfig and **AltPQuantEdgeMask** are derived based on the following variables: *DQUANT*, *DQUANTFRM*, *DQPROFILE*, *DQSBEDGE*, *DQDBEDGE*, and *DQBILEVEL* defined in the VC1 standard, as shown in the following table.

Definition of AltPQuantConfig and AltPQuantEdgeMask

Inputs						Outputs		Description
DQUANT	DQUANT FRM	DQ PROFILE	DQDB EDGE	DQSB EDGE	DQBI LEVEL	AltPQuant Config	AltPQuant EdgeMask	
0	-	-	-	-	-	00b	0000b	No **AltPQuant**
1	0	-	-	-	-	00b	0000b	No **AltPQuant**
1	1	11b	-	-	0	10b	0000b	All MBs are different with *MQDIFF* and *ABSMQ*
1	1	11b	-	-	1	11b	0000b	All MBs may switch with 1-bit *MQDIFF*
2	-	-	-	-	-	01b	1111b	All edge MBs
1	1	00b	-	-	-	01b	1111b	All edge MBs
1	1	01b	00b	-	-	01b	0011b	Left and top MBs
1	1	01b	01b	-	-	01b	0110b	Top and right MBs
1	1	01b	10b	-	-	01b	1100b	Right and bottom MBs
1	1	01b	11b	-	-	01b	1001b	Bottom and left MBs
1	1	10b	-	00b	-	01b	0001b	Left MBs
1	1	10b	-	01b	-	01b	0010b	Top MBs
1	1	10b	-	10b	-	01b	0100b	Right MBs
1	1	10b	-	11b	-	01b	1000b	Bottom MBs

MFD_VC1_SHORT_PIC_STATE

Intel HW does not use the MVMODE and MVMODE2 provided at the revised DXVA2 VC1 VLD interface, instead, HW will decode them directly from the bitstream picture header.

MFD_VC1_BSD_OBJECT

For VC1, a slice/picture is always started with MB x positon equal to 0. Hence, no need to include in the Object Command.

Handling Emulation Bytes

In general, VC1 BSD unit is capable of handling emulation prevention bytes. However, there is a corner case that requires host software's intervention. Host software needs to overwrite the emulation byte if it overlaps the macroblock layer decode and there is not enough information for the hardware to detect the emulation byte.

The emulation bytes might have an overlap between the picture states and the first macroblock data. If the emulation bytes are 0x00 **0x000x03** 0x00 and the macroblock data starts in the middle of byte1 (**0x00**), then the host software needs to overwrite the **0x03** byte location with the previous byte (**0x00**) and change the byte offset accordingly. The hardware wouldn't know what the 1st byte was and will miss this **0x03** removal.

VP8

This topic is currently under development.

MFX_VP8_STATISTICS - Encoder Only

Address offset			Name	Description	
0	31:0	32-bit	P0 Partition Bit Size	// Raw bit count per Partition	No Fulsim Validation
1	31:0	32-bit	P1 Partition Bit Size		No Fulsim Validation
2	31:0	32-bit	P2 Partition Bit Size		No Fulsim Validation
3	31:0	32-bit	P3 Partition Bit Size		No Fulsim Validation
4	31:0	32-bit	P4 Partition Bit Size		No Fulsim Validation
5	31:0	32-bit	P5 Partition Bit Size		No Fulsim Validation
6	31:0	32-bit	P6 Partition Bit Size		No Fulsim Validation
7	31:0	32-bit	P7 Partition Bit Size		No Fulsim Validation
8	31:0	32-bit	P8 Partition Bit Size		No Fulsim Validation
9	31:0	32-bit	P1-8 Partition Bit Size Sum	// This is raw BIT sum of Partition1-8, Not Byte sum.	No Fulsim Validation
					No Fulsim Validation
10	31:0	32-bit	Segment0 Total Bit Count	// Raw bit count per Segment	No Fulsim Validation
11	31:0	32-bit	Segment1 Total Bit Count		No Fulsim Validation
12	31:0	32-bit	Segment2 Total Bit Count		No Fulsim Validation
13	31:0	32-bit	Segment3 Total Bit Count		No Fulsim Validation
14	15:0	16-bit	Segment0 Num of MB	// Num of MB per Segment	No Fulsim Validation
	31:16	16-bit	Segment1 Num of MB		No Fulsim Validation

Address offset			Name	Description	
15	15:0	16-bit	Segment2 Num of MB		No Fulsim Validation
	31:16	16-bit	Segment3 Num of MB		No Fulsim Validation
16	31:0	32-bit	P0 Partition Bit Size before CPBAC	// Bin Count of Syntax Element before CPBAC	No Fulsim Validation
17	31:0	32-bit	P1 Partition Bit Size before CPBAC		No Fulsim Validation
18	31:0	32-bit	P2 Partition Bit Size before CPBAC		No Fulsim Validation
19	31:0	32-bit	P3 Partition Bit Size before CPBAC		No Fulsim Validation
20	31:0	32-bit	P4 Partition Bit Size before CPBAC		No Fulsim Validation
21	31:0	32-bit	P5 Partition Bit Size before CPBAC		No Fulsim Validation
22	31:0	32-bit	P6 Partition Bit Size before CPBAC		No Fulsim Validation
23	31:0	32-bit	P7 Partition Bit Size before CPBAC		No Fulsim Validation
24	31:0	32-bit	P8 Partition Bit Size before CPBAC		No Fulsim Validation
25	31:0	32-bit	Reserved		
26	31:0	32-bit	Reserved		
27	31:0	32-bit	Reserved		
28	31:0	32-bit	Reserved		
29	31:0	32-bit	Reserved		
30	31:0	32-bit	Reserved		
31	31:0	32-bit	Reserved		
32	31:0	32-bit	mb skip prob Total cnt	Total Number of MB when MBNoCoeffSkip == 1(if MBNoCoeffSkip == 0, this field =0) Please see foot notes 1.	
33	31:0	32-bit	mb skip prob cnt	Number of MB with MBSkip == 1 when MBNoCoeffSkip == 1. Please see foot notes 1.	
303-34	15:0		token_statistic count	Token Statistics Counters. Please see foot notes 2.	

Address offset			Name	Description	
	31:16	Reserved			

Programming Note	
Context:	MFX_VP8_STATISTICS - Encoder Only

1. MBSkip_Prob_Total_Cnt and MBSkip_Prob_Cnt are collected to generate MBSkipProbability. After bit packing, Hardware returns both MBSkip_Prob_Total_Cnt and MBSkip_Prob_Cnt. Optimal packing could be performed in subsequent pass using MBSkipProbability of

   round (256 * (MBSkip_Prob_Total_Cnt- MBSkip_Prob_Cnt) / MBSkip_Prob_Total_Cnt))

2. Token Statistics counters collects token statistics of particular plane (4), coefficient band(8), neighbor context(3) and tree node position(11) as described in WebM Spec. Out of the space of 1056 counters, there are 270 of which has high significant in compression efficiency and are chosen for statistics collection.

VP8 Encoder StreamOut Format

VP8 Encoder StreamOut Format			
DW0	0	1-bit	MBLevelIntraMBConformanceFlag
	1	1-bit	MBLevelInterMBConformanceFlag
	2	1-bit	MbRcFlag
	3	1-bit	MBSkip
	7:4	Reserved	
	15:8	8-bit	MbClock16
	23:16	8-bit	MbX
	31:24	8-bit	MbY
DW1	12:0	13-bit	MB_Total_BitCount
	15:13	Reserved	
	28:16	13-bit	MB_Residual_BitCount
	31:29	Reserved	
DW2	24:0	25-bit	Cbp
	31:25	Reserved	
DW3	15:0	Reserved	
	16	1-bit	CoeffClampStatus
	23:17	Reserved	
	28:24	5-bit	MBType5Bits
	29	1-bit	IntraMBFlag
	30	1-bit	LastMbFlag
	31	Reserved	

VP8 Common Commands

Following are VP8 Common Commands:

MFX_VP8_PIC_STATE

For VP8 HW PAK, there are four VP8 versions supported and their programming is shown in Table1 below.

Version	MC Filter Select	Chroma Full Pixel MC Filter Mode	DBLK FilterType	DBLK FilterLevel for Segment0
0	0	0	0	Any FilterLevel
1	1	0	1	Any FilterLevel
2	1	0	0	0
3	1	1	1	0

Table1: VP8 Version

MC Filter Select: MFX_VP8_PIC_STATE.DW2.Bit0

Chroma Full Pixel MC Filter Mode: MFX_VP8_PIC_STATE.DW2.Bit1

DBKL Filter Type: MFX_VP8_PIC_STATE.DW2.Bit4

DBLK Filter Level for Segment0: MFX_VP8_PIC_STATE.DW3.Bit5:0

1. Note that when multiple segment is enabled, if Segment0 DBLK Filter is programmed to 0, Segment1,2,3 DBLK Filter should be set to 0 as well.
2. Note that MFX_VP8_Encoder_CFG.BW22.Bit22:20 (Bitstream Format Version). This field is used for generating Uncompressed header only. It is not used to control any Filter.

VP8 Decoder Commands

Following are VP* Decoder Commands:

MFD_VP8_BSD_OBJECT

VP8 Encoder Commands

MFX_VP8_Encoder_CFG

MFX_VP8_BSP_BUF_BASE_ADDR_STATE

MFX_VP8_PAK_OBJECT

VP8 PAK Object inline data:

Inline Data Description - VP8 PAK OBJECT

Y_Mode for macroblock in non-B mode

0	DC_PRED
1	V_PRED
2	H_PRED
3	TM_PRED
4	B_PRED
5	NEARESTMV
6	NEARMV
7	ZEROMV
8	NEWMV
9	SPLITMV

2 B mode

0	B_DC_PRED
1	B_TM_PRED
2	B_VE_PRED
3	B_HE_PRED
4	B_LD_PRED
5	B_RD_PRED
6	B_VR_PRED
7	B_VL_PRED
8	B_HD_PRED
9	B_HU_PRED
10	SPLIT_LEFT
11	SPLIT_ABOVE
12	SPLIT_ZERO
13	SPLIT_NEW

VP8 Bitstream Formats

There are the following restrictions on the input compressed bitstream format:

1. No Zero Coefficient coded before EOB;
2. No zero coefficient coded, only EOB coded if all subblock coefficients are 0's.

JPEG and MJPEG

JPEG Decoder Commands

Following are JPEG Decoder Commands:

MFD_JPEG_BSD_OBJECT

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.
MFX_JPEG_PIC_STATE

For JPEG decoding, the following program note is informative.

For **Rotation**, it is important to note that rotation of 90 or 270 degrees also requires exchanging **FrameWidthInBlksMinus1** with **FrameHeightInBlksMinus1** in the command. In addition, the rotation of 90 or 270 degrees also requires transportation of the quantization matrix will be transposed into the position (y, x).

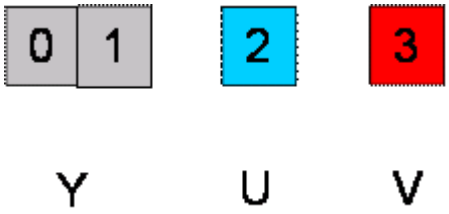
Chroma type is determined by the values of horizontal and vertical sampling factors of the components (*Hi* and Vi where *i* is a component id) in the Frame header as shown in the following table.

	H1	H2	H3	V1	V2	V3
0: YUV400	r	Not available	Not available	r	Not available	Not available
1: YUV420	2	1	1	2	1	1
2: YUV422H_2Y	2	1	1	1	1	1
3: YUV444	1	1	1	1	1	1
4: YUV411	4	1	1	1	1	1
5: YUV422V_2Y	1	1	1	2	1	1
6: YUV422H_4Y	2	1	1	2	2	2
7: YUV422V_4Y	2	2	2	2	1	1

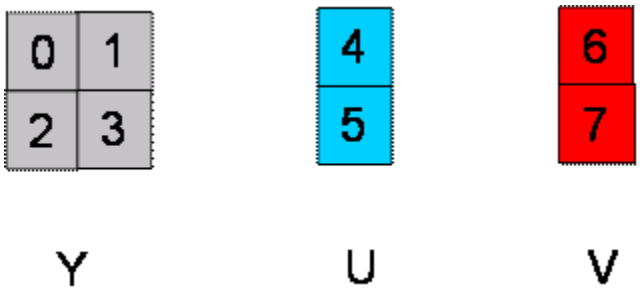
For YUV400, the value of *V1* can be 1, 2, or 3 and will be same as the value of *H1*, and the Minimum coded unit (MCU) is one 8x8 block. For the other chroma formats, if non-interleaved data, the MCU is one 8x8 block. For interleaved data, the MCU is the sequence of block units defined by the sampling factors of the components.

For example, the following figures show the MCU structures of interleaved data and the decoding order of blocks in the MCU:

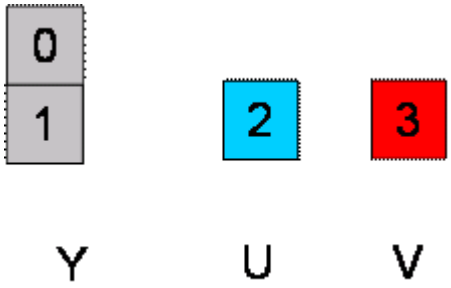
422H_2Y



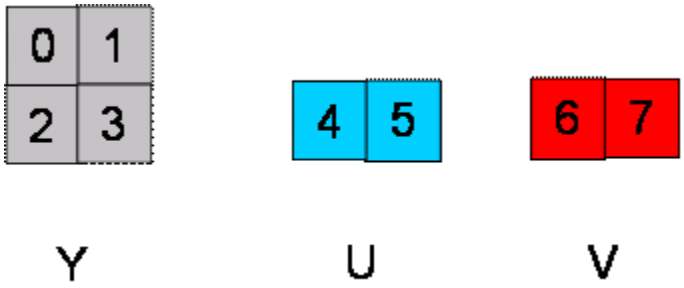
422H_4Y



422V_2Y

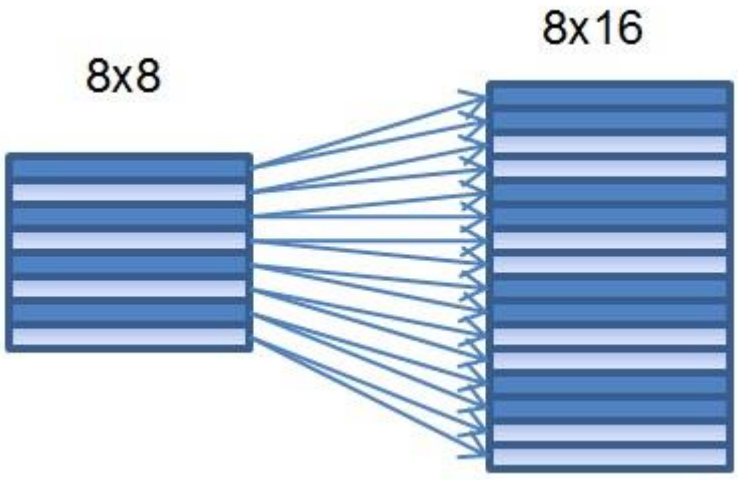


422V_4Y

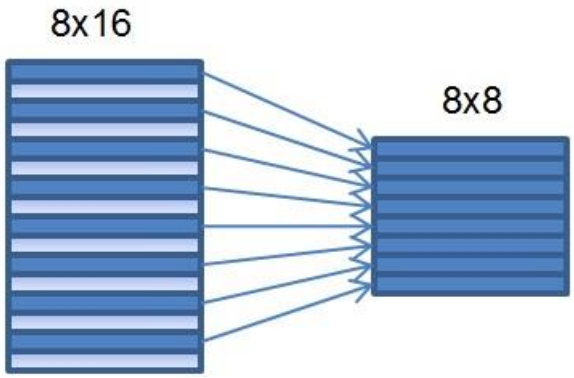
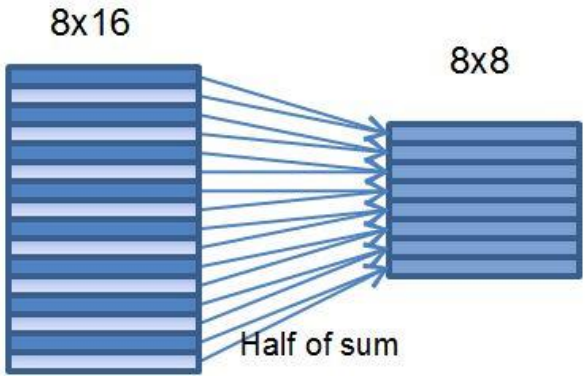


If picture width *X* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of columns to complete the right-most sample blocks. If the component is to be interleaved, the decoding process needs to extend the number of samples by one or more additional blocks so that the number of blocks is an integer multiple of *Hi*. In other words, "The number of blocks in width" in the table should be an integer multiple of (8x*H1*). Similarly, if picture height *Y* in the Frame header is not a multiple of 8, the decoding process needs to extend the number of lines to complete bottom-most block-row. If the component is to be interleaved, the decoding process also needs to extend the number of lines by one or more additional block-rows so that the number of block-row is an integer multiple of (8x*V1*). For example, if non-interleaved YUV411 with *X*=270, then "The number of blocks in width" shall be (270 + 7) / 8 = 34, where "/" is integer division. Therefore, **FrameWidthInBlksMinus1** is set to 33. However, for interleaved data, "The number of blocks in width" shall be ((270 + 31) / 32) x 4 = 36. Therefore, **FrameWidthInBlksMinus1** is set to 35.

VertUpSamplingEnb is used to convert an input chroma420 to an output chroma422 in the surface format of YUY2 or UYVY. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV420, and OutputFormatYUV should be set to YUY2 or UYVY. Vertical 2:1 up-sampling is only applied to chroma blocks where each line of 8x8 block pixels is replicated to make 8x16 U/V blocks. For example:



VertDownSamplingEnb is used to convert an input chroma422 to an output chroma420 in the surface format NV12. To enable this flag, the input should be interleaved Scan, InputFormatYUV should be set to YUV422H_2Y or YUV422H_4Y, and OutputFormatYUV should be set to NV12. Combined with AvgDownSampling flag, the following table and figures show the down-sampling methods.

VertDownSamplingEnb	AvgDownSampling	Down-Sampling Methods
0	0 or 1	No down-sampling.
1	0	Drop every other line: 
1	1	Average vertically neighboring two pixels: 

The recent history forJPEG Decoder Commands are described in the following:

- If the InputFormat is YUV400 or YUV444 or YUV411, then output cannot be NV12, YUY2 or UYVY, it has to be planar. But for 420 and 422 InputFormat, there's a choice of having Planar, NV12, YUY2 or UYVY OutputFormat. And the surface state should be programmed accordingly.
- Refer "Output Format YUV" field for more details.

MFX_JPEG_HUFF_TABLE_STATE

JPEG Encoder Commands

Following are JPEG Encoder Commands:

MFX_JPEG_PIC_STATE command is used for both encoding and decoding. Note the duplicate bits and the "Exists If" rows that specify what the bits represent for Encoder and Decoder.
MFX_JPEG_PIC_STATE

Programming Note: For completion of partial MCUs in JPEG encoding, it is important to note the following:

If the image's dimensions are not an exact multiple of the MCU size, the encoded data should include padding to round up to the next complete MCU, which is called completion of partial MCU. If the number of lines is not aligned with MCU structure (not a multiple of MCU size, i.e. 8, 16, 32), the encoding process needs to extend the number of lines to complete the bottom-most MCU-row. Similarly, if the number of samples per line is not aligned with MCU structure, the encoding process needs to extend the number of columns to complete the right-most sample MCUs. JPEG standard recommends that any incomplete MCUs be completed by replication of the right-most column and the bottom line of each component Y, U, and V.

The following equations are used to set the command for encoding partial MCUs.

FrameWidthInBlksMinus1 = $(((X + (H_1 * 8 - 1)) / (H_1 * 8)) * H_1) - 1$

FrameHeightInBlksMinus1 = $(((Y + (V_1 * 8 - 1)) / (V_1 * 8)) * V_1) - 1$

```
For YUV400,
    PixelsInHoriLastMCU = X % 8
    PixelsInVertLastMCU = Y % 8

For YUV420,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 16 if Y % 2 = 0, ((Y % 16) + 1) % 16 if Y % 2 = 1

For YUV422H_2Y,
    PixelsInHoriLastMCU = X % 16 if X % 2 = 0, ((X % 16) + 1) % 16 if X % 2 = 1
    PixelsInVertLastMCU = Y % 8

    X: the number of samples per line in Y-image

    Y: the number of lines in Y-image

    H1: horizontal sampling factor of Y-image in the Frame header

    V1:  vertical sampling factor of Y-image in the Frame header
```

Note that PixelsInHoriLastMCU=0 does not mean the num of pixels in the right-most MCUs = 0, but does mean that the right-most MCUs are fully filled with pixels, i.e., not a partial MCU.

For example, for input image dimension 17x26 pixels and an interleaved Scan, the following equations and the table show how to set the command for each OutputMcuStructure.

	YUV400	YUV420	YUV422H_2Y
MCU size of Y	8x8	16x16	16x8

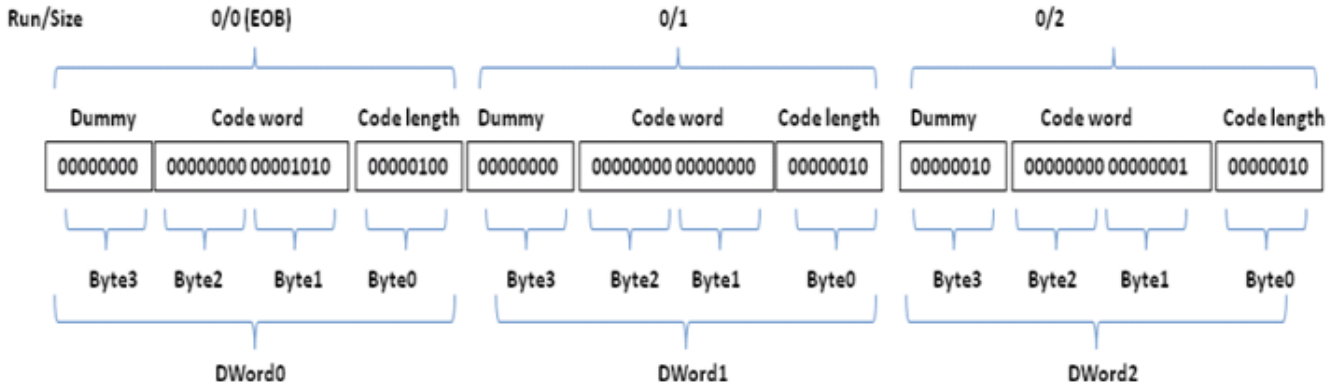
	YUV400	YUV420	YUV422H_2Y
MCU size of U and V	8x8	8x8	8x8
H_1 and V_1	1, 1	2, 2	2, 1
FrameWidthInBlksMinus1	2	3	3
FrameHeightInBlksMinus1	3	3	3
PixelsInHoriLastMCU	1	2	2
PixelsInVertLastMCU	2	10	2

MFC_JPEG_SCAN_OBJECT

MFC_JPEG_ HUFF_TABLE_STATE
The JPEG standard Table K.5 shows the real table of code length and code word as follows:

Run/Size	Code length	Code word
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
0/4	4	1011
0/5	5	11010
0/6	7	1111000
0/7	8	11111000
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011

It is not necessary to send Run/size in the command as driver will send the increasing order of run/size. Each symbol aligns to a DWord with the following byte structure. Each DWord (a group of 4 bytes) contains Byte0 for Code length, Byte1 and Byte2 for Code word, and Byte3 for dummy.



Driver will program to always send 12 pairs of Code length and Code Word in DC coefficient table and 162 pairs in AC coefficient table. When a Huffman table contains valid full entries of Run/Size, all the Code word and Code length will not be zero. If a Huffman table is customized or optimized, the table can contain smaller set of Code length and Code Word, i.e., the number of entries of the real Huffman table will be less than 12 for DC, or less than 162 for AC. For the customized Huffman table, driver will set the missing entry (Run/Size) to Code length = 0 and Code word = 0.

MFX_PAK_INSERT_OBJECT

More Decoder and Encoder

MFD IT Mode Decode Commands

These are decoder-only commands to support the IT-mode specified in DXVA interface.

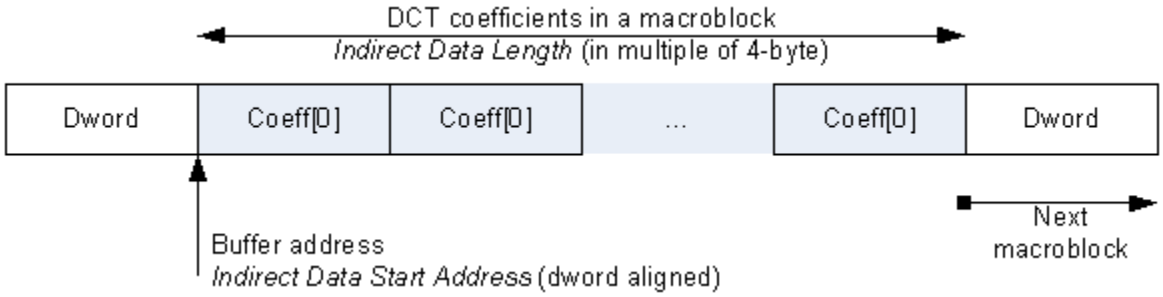
MFD_IT_OBJECT

Common Indirect IT-COEFF Data Structure

Transform-domain residual data block in AVC-IT, VC1-IT and MPEG2-IT mode follows the same data structure.

The indirect IT-COEFF data start address in MFD_IT_OBJECT command specifies the doubleword aligned address of the first non-zero DCT coefficient of the first block of the macroblock. Only the non-zero coefficients are present in the data buffer and they are packed in the 8x8 block sequence of Y0, Y1, Y2, Y3, Cb4 and Cr5, as shown in *Common Indirect IT-COEFF Data Structure*. When an 8x8 block is further subdivided into 4x4 subblocks, the coefficients, if present, are organized in the subblock order. The smallest subblock division is referred to as a **transform block**. The indirect IT-COEFF data length in the command includes all the non-zero coefficients for the macroblock. It must be doubleword aligned.

Structure of the IDCT Compressed Data Buffer



Each non-zero coefficient in the indirect data buffer is contained in a doubleword-size data structure consisting of the coefficient index, end of block (EOB) flag and the fixed-point coefficient value in 2's compliment form. As shown in *Common Indirect IT-COEFF Data Structure*, *index* is the row major 'raster' index of the coefficient **within a transform block** (*please note that it is not converted to 8x8 block basis*). A coefficient is a 16-bit value in 2's complement.

Structure of a transform-domain residue unit

DWord	Bit	Description
0	31:16	**Transform-Domain Residual (coefficient) Value.** This field contains the value of the non-zero transform-domain residual data in 2's compliment.
	15:7	Reserved: MBZ
	6:1	**Index.** This field specifies the raster-scan address (raw address) of the coefficient within the transform block. For a coefficient at Cartesian location (row, column) = (y, x) in a transform block of width W, Index is equal to (y * W + x). For example, coefficient at location (row, column) = (0, 0) in a 4x4 transform block has an index of 0; that at (2, 3) has an index of 2*4 + 3 = 11. The valid range of this field depends on the size of the transform block. Format = U6 Range = [0, 63]
	0	**EOB (End of Block).** This field indicates whether the transform-domain residue is the last one of the current transform block.

Allowed transform block dimensions per coding standard

Transform Block Dimension	AVC	VC1	MPEG2
8x8	Yes	Yes	Yes
8x4	No	Yes	No
4x8	No	Yes	No
4x4	Yes	Yes	No

For AVC, there is intra16x16 mode, in which the DC Luma coefficients of all 4x4 sub-blocks within the current MB are sent separately in its own 4x4 Luma block. As such, only 15 coefficients remains in each of the 16 4x4 Luma blocks.

Inline Data Description in AVC-IT Mode

The Inline Data includes all the required MB decoding states, extracted primarily from the Slice Data, MB Header and their derivatives. It provides information for the following operations:

1. Inverse Quantization
2. Inverse Transform
3. Intra and inter-Prediction decoding operations
4. Internal error handling

IT Mode supports only packed MV data as specified in the DXVA spec.

These state/parameter values may subject to change on a per-MB basis, and must be provided in each MFD_IT_OBJECT command. The values set for these variables are retained internally, until they are reset by hardware Asynchronous Reset or changed by the next MFC_AVC_PAK_OBJECT command.

The inline data has been designed to match the DXVA 2.0, with the exception of the starting byte (DW0:0-7) and the ending dword (DW7:0-31).

The Deblocker Filter Control flags (FilterInternalEdgesFlag, FilterTopMbEdgeFlag and FilterLeftMbEdgesFlag) are generated by H/W, which are depending on MbaffFrameFlag, CurrMbAddr, PicWidthInMbs and disable_deblocking_filter_idc states.

Current MB [x,y] address is not sent, it is assumed that the H/W will keep track of the MB count and current MB position internally.

DWord	Bit	Description
0	31:24	**MvQuantity** Specify the number of MVs (in unit of motion vector, 4 bytes each) to be fetched for motion compensation operation. Decoder IT mode only supports packed MV format (DXVA). This field specifies the exact number of MVs present for the current MB. For a P-Skip MB, there is still 1 MV being sent (Skip MV is sent explicitly); for a B-Direct/Skip MB, there are 2 MVs being sent. For an Intra-MB, MvQuantity is set to 0. MvQuantity = 0, signifies there is no MV indirect data for the current MB. This field must be set in consistent with **Indirect MV Data Length**, so as not to exceed its bound Unsigned.
	23:20	Reserved MBZ (DXVA)
	19	**DcBlockCodedYFlag** 1 – the 4x4 DC-only Luma sub-block of the Intra16x16 coded MB is present; it is still possible that all DC coefficients are zero. 0 – no 4x4 DC-only Luma sub-block is present; either not in Intra16x16 MB mode or all DC coefficients are zero.
	18	**DcBlockCodedCbFlag** For 4:2:0 case : 1 – the 2x2 DC-only Chroma Cb sub-block of all coded MB (any type) is present; it is still possible that all DC coefficients are zero. 0 – no 2x2 DC-only Chroma Cb sub-block is present; all DC coefficients are zero.
	17	**DcBlockCodedCrFlag** For 4:2:0 case : 1 – the 2x2 DC-only Chroma Cr sub-block of all coded MB (any type) is present; it

DWord	Bit	Description
		is still possible that all DC coefficients are zero. 0 – no 2x2 DC-only Chroma Cr sub-block is present; all DC coefficients are zero.
	16	Reserved MBZ (DXVA)
	15	**Transform8x8Flag** 0: indicates the current MB is coded with 4x4 transform and therefore the luma residuals are presented in 4x4 blocks. 1: indicates the current MB is coded with 8x8 transform and therefore the luma residuals are presented in 8x8 blocks. Same as the transform_szie_8x8_flag syntax element in AVC spec.
	14	**MbFieldFlag** This field specifies whether current macroblock is coded as a field or frame macroblock in MBAFF mode. 1 = Field macroblock 0 = Frame macroblock This field is exactly the same as FIELD_PIC_FLAG syntax element in non-MBAFF mode. Same as the mb_field_decoding_flag syntax element in AVC spec.
	13	**IntraMbFlag** This field specifies whether the current macroblock is an Intra (I) macroblock. 0 – not an intra MB 1 – is an intra MB I_PCM is considered as Intra MB. For I-picture MB (IntraPicFlag =1), this field must set to 1. This flag must be set in consistent with the interpretation of MbType (inter or intra modes).
	12:8	**MbType** This field carries the Macroblock Type. The meaning depends on IntraMbFlag. If IntraMbFlag is 1, this field is the intra macroblock type as defined in MbType definition for Intra Macroblock . If IntraMbFlag is 0, this field is the inter macroblock type as defined in the first two columns of MbType definition for Inter Macroblock (and MbSkipflag = 0). All macroblock types in a P Slice are mapped into the corresponding types in a B Slice. Skip and Direct modes are converted into its corresponding processing

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>modes.
Programming note: It is exactly matched with that of DXVA 2.0.</td></tr>
<tr><td></td><td>7</td><td>FieldMbPolarityFlag
This field indicates the field polarity of the current macroblock.
Within a MbAff frame picture, this field may be different per macroblock and is set to 1 only for the second macroblock in a MbAff pair if FieldMbFlag is set. Otherwise, it is set to 0.
Within a field picture, this field is set to 1 if the current picture is the bottom field picture. Otherwise, it is set to 0. It is a constant for the whole field picture.
This field is only valid for MBAFF frame picture. It is reserved and set to 0 for a progressive frame picture or a field picture.
0 = Current macroblock is a field macroblock from the top field (first in a MBAFF pair)
1 = Current macroblock is a field macroblock from the bottom field (second in a MBAFF pair)</td></tr>
<tr><td></td><td>6</td><td>IsLastMB
1 – the current MB is the last MB in the current Slice
0 – the current MB is not the last MB in the current Slice</td></tr>
<tr><td></td><td>5-4</td><td>Reserved MBZ (Intel encoder)</td></tr>
<tr><td></td><td>3:0</td><td>Reserved MBZ (DXVA Decoder)</td></tr>
<tr><td>1</td><td>31:16</td><td>CbpY[bit 15:0] (Coded Block Pattern Y)
For 4x4 sub-block (when Transform8x8flag = 0 or in intra16x16) :
block in intra16x16) in a MB. The 4x4 Luma sub-blocks are numbered as
<table>
<tr><td>blk0</td><td>1</td><td>4</td><td>5</td><td></td><td>14</td><td>11</td><td>10</td></tr>
<tr><td>blk2</td><td>3</td><td>6</td><td>7</td><td>bit 13</td><td>12</td><td>9</td><td>8</td></tr>
<tr><td>blk8</td><td>9</td><td>12</td><td>13</td><td>bit 7</td><td>6</td><td>3</td><td>2</td></tr>
<tr><td>blk10</td><td>11</td><td>14</td><td>15</td><td>bit 5</td><td>4</td><td>1</td><td>0</td></tr>
</table>
The cbpY bit assignment is cbpY bit [15 – X] for sub-block_num X.
For 8x8 block (when Transform8x8flag = 1)</td></tr>
</table>

DWord	Bit	Description
		Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored. The 8x8 Luma blocks are numbered as <table><tr><td>blk0</td><td>1</td><td>bit 3</td><td>2</td></tr><tr><td>blk2</td><td>3</td><td>bit 1</td><td>0</td></tr></table> The cbpY bit assignment is cbpY bit [3 – X] for block_num X. 0 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is not present (because all coefficient values are zero) 1 in a bit – indicates the corresponding 8x8 block or 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).
	15:8	**VertOrigin (Vertical Origin).** This field specifies the vertical origin of current macroblock in the destination picture in units of macroblocks. For field macroblock pair in MBAFF frame, the vertical origins for both macroblocks should be set as if they were located in corresponding field pictures. For example, for field macroblock pair originated at (16, 64) pixel location in an MBAFF frame picture, the Vertical Origin for both macroblocks should be set as 2 (macroblocks). Whether the current macroblock is the first/second (top/bottom) in a MBAFF pair is specified by **FieldMbPolarityFlag.** The macroblocks with (**VertOrigin, HorzOrigin**) must be delivered in the strict order as coded in the bitstream (raster order for progressive frame or field pictures and MBAFF pair order for MBAFF pictures). No gap is allowed. Otherwise, hardware behavior is undefined. Format = U8 in unit of macroblock.
	7:0	**HorzOrigin (Horizontal Origin).** This field specifies the horizontal origin of current macroblock in the destination picture in units of macroblocks. Format = U8 in unit of macroblock.
2	31:16	**CbpCr (Coded Block Pattern Cr 4:2:0-only)** Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored (only valid for 4:2:2 and 4:4:4). The 4x4 Chroma Cr sub-blocks are numbered as <table><tr><td>blk0</td><td>1</td><td>bit 3</td><td>2</td></tr><tr><td>blk2</td><td>3</td><td>bit 1</td><td>0</td></tr></table> The cbpCr bit assignment is cbpCr bit [3 – X] for sub-block_num X.

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td></td><td></td><td>0 in a bit – indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)

1 in a bit – indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).

For monochrome, this field is ignored.</td></tr>
<tr><td></td><td>15-0</td><td>CbpCb (Coded Block Pattern Cb 4:2:0-only)

Only the lower 4 bits [3:0] are valid; the remaining upper bits [15:4] are ignored (only valid for 4:2:2 and 4:4:4). The 4x4 Chroma Cb sub-blocks are numbered as
<table><tr><td>blk0</td><td>1</td><td>bit 3</td><td>2</td></tr><tr><td>blk2</td><td>3</td><td>bit 1</td><td>0</td></tr></table>
The cbpCb bit assignment is cbpCb bit [3 – X] for sub-block_num X.

0 in a bit – indicates the corresponding 4x4 sub-block is not present (because all coefficient values are zero)

1 in a bit – indicates the corresponding 4x4 sub-block is present (although it is still possible to have all its coefficients be zero – bad coding).

For monochrome, this field is ignored.</td></tr>
<tr><td>3</td><td>31:24</td><td>Reserved MBz</td></tr>
<tr><td></td><td>23:16</td><td>QpPrimeCr

Driver is responsible for deriving the QpPrimeCr from QpPrimeY.

For 8-bit pixel data, QpCr is the same as QpPrimeCr, and it takes on a value in the range of 0 to 51, positive integer.</td></tr>
<tr><td></td><td>15:8</td><td>QpPrimeCb

Driver is responsible for deriving the QpPrimeCb from QpPrimeY.

For 8-bit pixel data, QpCb is the same as QpPrimeCb, and it takes on a value in the range of 0 to 51, positive integer.</td></tr>
<tr><td></td><td>7:0</td><td>QpPrimeY

This is the per-MB QP value specified for the current MB.

For 8-bit pixel data, QpY is the same as QpPrimeY, and it takes on a value in the range of 0 to 51, positive integer.</td></tr>
</table>

DWord	Bit	Description
4 to 6	31:0 Each	For intra macroblocks, definition of these fields are specified in Inline data subfields for an Intra Macroblock For inter macroblocks, definition of these fields are specified in Inline data subfields for an Inter Macroblock

Indirect Data Format in AVC-IT Mode

Indirect data in AVC-IT mode consist of Motion Vectors, Transform-domain Residue (Coefficient) and ILDB control data. All three data records have variable size. Size of each Motion Vector record is determined by the MvQuantity value as shown in *Indirect Data Format in AVC-IT Mode*. ILDB control record is fixed at the same size for all MBs in a picture. Coefficient data record is variable size per MB, since it may only consist of non-zero coefficients.

Each MV is represented in 4 bytes, in the form of

- Lower 2 bytes : horizontal MVx component in q-pel units
- Upper 2 bytes : vertical MVy component in q-pel units
- Integer distance is measured in unit of samples in the frame or field grid position.
- Chroma MVs are not sent and are derived in the H/W.

Indirect MV record size in AVC-IT mode

Macroblock Type	MVQuant
BP_L0_16x16	1
B_L1_16x16	1
B_Bi_16x16	2
BP_L0_L0_16x8	2
BP_L0_L0_8x16	2
B_L1_L1_16x8	2
B_L1_L1_8x16	2
B_L0_L1_16x8	2
B_L0_L1_8x16	2
B_L1_L0_16x8	2

Macroblock Type	MVQuant
B_L1_L0_8x16	2
B_L0_Bi_16x8	3
B_L0_Bi_8x16	3
B_L1_Bi_16x8	3
B_L1_Bi_8x16	3
B_Bi_L0_16x8	3
B_Bi_L0_8x16	3
B_Bi_L1_16x8	3
B_Bi_L1_8x16	3
B_Bi_Bi_16x8	4
B_Bi_Bi_8x16	4
BP_8x8	Sum

For macroblock type of BP_8x8, MvQuant takes the sum of value MvQ[i] of the four individual 8x8 sub macroblocks.

SubMbShape[i]	SubMbPredMode[i]	Description	MvQ[i]
0	0	BP_L0_8x8	1
0	1	B_L1_8x8	1
0	2	B_BI_8x8	2
1	0	BP_L0_8x4	2
1	1	B_L1_8x4	2
1	2	B_BI_8x4	4
2	0	BP_L0_4x8	2
2	1	B_L1_4x8	2
2	2	B_BI_4x8	4
3	0	BP_L0_4x4	4
3	1	B_L1_4x4	4
3	2	B_BI_4x4	8

Indirect data Deblocking Filter Control block in AVC-IT mode:

AVC Deblocker Control Data record has a fixed size for each MB in a picture and is 48 bytes or 12 Dwords in size.

DWord	Bit	Description
0	31:24	Reserved : MBZ (DXVA Decoder)
	23	**FilterTopMbEdgeFlag**
	22	**FilterLeftMbEdgeFlag**
	21	**FilterInternal4x4EdgesFlag**
	20	**FilterInternal8x8EdgesFlag**
	19	**FieldModeAboveMbFlag**
	18	**FieldModeLeftMbFlag**
	17	**FieldModeCurrentMbFlag**
	16	**MbaffFrameFlag** (DXVA Decoder reserved bit)
	15:8	**VertOrigin** Current MB y position (address)
	7:0	**HorzOrigin** Current MB x position (address)
1	31:30	**bS_h13** 2-bit boundary strength for internal top horiz 4-pixel edge 3
	29:28	**bS_h12** 2-bit boundary strength for internal top horiz 4-pixel edge 2
	27:26	**bS_h11** 2-bit boundary strength for internal top horiz 4-pixel edge 1
	25:24	**bS_h10** 2-bit boundary strength for internal top horiz 4-pixel edge 0
	23:22	**bS_v33** 2-bit boundary strength for internal right vert 4-pixel edge 3
	21:20	**bS_v23** 2-bit boundary strength for internal right vert 4-pixel edge 2
	19:18	**bS_v13** 2-bit boundary strength for internal right vert 4-pixel edge 1
	17:16	**bS_v03** 2-bit boundary strength for internal right vert 4-pixel edge 0
	15:14	**bS_v32** 2-bit boundary strength for internal mid vert 4-pixel edge 3
	13:12	**bS_v22** 2-bit boundary strength for internal mid vert 4-pixel edge 2
	11:10	**bS_v12** 2-bit boundary strength for internal mid vert 4-pixel edge 1
	9:8	**bS_v02** 2-bit boundary strength for internal mid vert 4-pixel edge 0
	7:6	**bS_v31** 2-bit boundary strength for internal left vert 4-pixel edge 3
	5:4	**bS_v21** 2-bit boundary strength for internal left vert 4-pixel edge 2
	3:2	**bS_v11** 2-bit boundary strength for internal left vert 4-pixel edge 1
	1:0	**bS_v01** 2-bit boundary strength for internal left vert 4-pixel edge 0
2	31:28	**bS_v30_0** 4-bit boundary strength for Left0 4-pixel edge 3 (MSbit is wasted)
	17:24	**bS_v20_0** 4-bit boundary strength for Left0 4-pixel edge 2 (MSbit is wasted)
	23:20	**bS_v10_0** 4-bit boundary strength for Left0 4-pixel edge 1 (MSbit is wasted)
	19:16	**bS_v00_0** 4-bit boundary strength for Left0 4-pixel edge 0 (MSbit is wasted)
	15:14	**bS_h33** 2-bit boundary strength for internal bot horiz 4-pixel edge 3
	13:12	**bS_h32** 2-bit boundary strength for internal bot horiz 4-pixel edge 2

DWord	Bit	Description
	11:10	**bS_h31** 2-bit boundary strength for internal bot horiz 4-pixel edge 1
	9:8	**bS_h30** 2-bit boundary strength for internal bot horiz 4-pixel edge 0
	7:6	**bS_h23** 2-bit boundary strength for internal mid horiz 4-pixel edge 3
	5:4	**bS_h22** 2-bit boundary strength for internal mid horiz 4-pixel edge 2
	3:2	**bS_h21** 2-bit boundary strength for internal mid horiz 4-pixel edge 1
	1:0	**bS_h20** 2-bit boundary strength for internal mid horiz 4-pixel edge 0
3	31:28	**bS_h03_0** 4-bit boundary strength for Top0 4-pixel edge 3 (MSbit is wasted)
	27:24	**bS_h02_0** 4-bit boundary strength for Top0 4-pixel edge 2 (MSbit is wasted)
	23:20	**bS_h01_0** 4-bit boundary strength for Top0 4-pixel edge 1 (MSbit is wasted)
	19:16	**bS_h00_0** 4-bit boundary strength for Top0 4-pixel edge 0 (MSbit is wasted)
	15:12	**bS_v03** 4-bit boundary strength for Left1 4-pixel edge 3 (MSbit is wasted)
	11:8	**bS_v02** 4-bit boundary strength for Left1 4-pixel edge 2 (MSbit is wasted)
	7:4	**bS_v01** 4-bit boundary strength for Left1 4-pixel edge 1 (MSbit is wasted)
	3:0	**bS_v00** 4-bit boundary strength for Left1 4-pixel edge 0 (MSbit is wasted)
4	31:24	**bIndexBinternal_Y** Internal index B for Y
	23:16	**bIndexAinternal_Y** Internal index A for Y
	15:12	**bS_h03_1** 4-bit boundary strength for Top1 4-pixel edge 3 (MSbit is wasted)
	11:8	**bS_h02_1** 4-bit boundary strength for Top1 4-pixel edge 2 (MSbit is wasted)
	7:4	**bS_h01_1** 4-bit boundary strength for Top1 4-pixel edge 1 (MSbit is wasted)
	3:0	**bS_h00_1** 4-bit boundary strength for Top1 4-pixel edge 0 (MSbit is wasted)
5	31:24	**bIndexBleft1_Y**
	23:16	**bIndexAleft1_Y**
	15:8	**bIndexBleft0_Y**
	7:0	**bIndexAleft0_Y**
6	31:24	**bIndexBtop1_Y**
	23:16	**bIndexAtop1_Y**
	15:8	**bIndexBtop0_Y**
	7:0	**bIndexAtop0_Y**
7	31:24	**bIndexBleft0_Cb**
	23:16	**bIndexAleft0_Cb**
	15:8	**bIndexBinternal_Cb**
	7:0	**bIndexAinternal_Cb**
8	31:24	**bIndexBtop0_Cb**
	23:16	**bIndexAtop0_Cb**
	15:8	**bIndexBleft1_Cb**
	7:0	**bIndexAleft1_Cb**

DWord	Bit	Description
9	31:24	**bIndexBinternal_Cr**
	23:16	**bIndexAinternal_Cr**
	15:8	**bIndexBtop1_Cb**
	7:0	**bIndexAtop1_Cb**
10	31:24	**bIndexBleft1_Cr**
	23:16	**bIndexAleft1_Cr**
	15:8	**bIndexBleft0_Cr**
	7:0	**bIndexAleft0_Cr**
11	31:24	**bIndexBtop1_Cr**
	23:16	**bIndexAtop1_Cr**
	15:8	**bIndexBtop0_Cr**
	7:0	**bIndexAtop0_Cr**

Inline Data Description in VC1-IT Mode

<table>
<tr><th>DWord</th><th>Bits</th><th>Description</th></tr>
<tr><td>+0</td><td>31:28</td><td>MvFieldSelect. A bit-wise representation indicating which field in the reference frame is used as the reference field for current field. It’s only used in decoding interlaced pictures.

This field is valid for non-intra macroblock only.

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>28</td><td>Forward predict of current frame/field or TOP field of interlace frame, or block 0 in 4MV mode.</td></tr>
<tr><td>29</td><td>Backward predict of current frame/field or TOP field of interlace frame, or forward predict for block 1 in 4MV mode.</td></tr>
<tr><td>30</td><td>Forward predict of BOTTOM field of interlace frame, or block 2 in 4MV mode.</td></tr>
<tr><td>31</td><td>Backward predict of BOTTOM field of interlace frame, or forward predict for block 3 in 4MV mode.</td></tr>
</table>

Each corresponding bit has the following indication.

0 = The prediction is taken from the top reference field.

1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>27</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>26</td><td>MvFieldSelectChroma . This field specifies the polarity of reference field for chroma blocks when their motion vector is derived in Motion4MV mode for interlaced (field) picture.

Non-intra macroblock only. This field is derived from MvFieldSelect.</td></tr>
</table>

DWord	Bits	Description
		0 = The prediction is taken from the top reference field. 1 = The prediction is taken from the bottom reference field.
	25:24	**MotionType – Motion Type** For frame picture, a macroblock may only be either 00 or 10. For interlace picture, a macroblock may be of any motion types. It can be 01 if and only if DctType is 1. This field is 00 if and only if IntraMacroblock is 1. 00 = Intra 01 = Field Motion. 10 = Frame Motion or no motion. Others = Reserved.
	23	**Reserved.** MBZ
	22	**MvSwitch.** This field specifies whether the prediction needs to be switched from forward to backward or vice versa for single directional prediction for top and bottom fields of interlace frame B macroblocks. 0 = No directional prediction switch from top field to bottom field 1 = Switch directional prediction from top field to bottom field
	21	**DctType.** This field specifies whether the residual data is coded as field residual or frame residual for interlaced picture. This field can be 1 only if MotionType is 00 (intra) or 01 (field motion). For progressive picture, this field must be set to '0', i.e. all macrobalcoks are frame macroblock. 0 = Frame residual type. 1 = Field residual type.
	20	**OverlapTransform.** This field indicates whether overlap smoothing filter should be performed on I-block boundaries. 0 = No overlap smoothing filter. 1 = Overlap smoothing filter performed.
	19	**Motion4MV.** This field indicates whether current macroblock a progressive P picture uses 4 motion vectors, one for each luminance block. It's only valid for progressive P-picture decoding. Otherwise, it is reserved and MBZ. For example, with MotionForward is 0, this field must also be set to 0. 0 = 1MV-mode.

DWord	Bits	Description
		1 = 4MV-mode.
	18	**MotionBackward.** This field specifies whether the backward motion vector is active for B-picture. This field must be 0 if Motion4MV is 1 (no backward motion vector in 4MV-mode). 0 = No backward motion vector. 1 = Use backward motion vector(s).
	17	**MotionForward.** This field specifies whether the forward motion vector is active for P and B pictures. 0 = No forward motion vector. 1 = Use forward motion vector(s).
	16	**IntraMacroblock.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used). For field motion, this field indicates whether the top field of the macroblock is coded as intra. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**LumaIntra8x8Flag – Luma Intra 8x8 Flag** This field specifies whether each of the four 8x8 luminance blocks are intra or inter coded when Motion4MV is set to 4MV-Mode. Each bit corresponds to one block. “0” indicates the block is inter coded and ‘1’ indicates the block is intra coded. When Motion4MV is not 4MV-Mode, this field is reserved and MBZ. Bit 15: Y0 Bit 14: Y1 Bit 13: Y2 Bit 12: Y3
	11:6	**CBP - Coded Block Pattern** This field specifies whether the 8x8 residue blocks in the macroblock are present or not. Each bit corresponds to one block. “0” indicates residue block isn’t present, “1” indicates residue block is present. Note: For each block in an intra-coded macroblock or an intra-coded block in a P macroblock in 4MV-Mode, the corresponding CBP must be 1. Subsequently, there must be at least one coefficient (this coefficient might be zero) in the indirect data buffer associated with the bock (i.e. residue block must be present).

DWord	Bits	Description
		Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 Bit 6: Cr5
	5	**ChromaIntraFlag - Derived Chroma Intra Flag** This field specifies whether the chroma blocks should be treated as intra blocks based on motion vector derivation process in 4MV mode. 0 = Chroma blocks are not coded as intra. 1 = Chroma blocks are coded as intra
	4	**LastRowFlag – Last Row Flag** This field indicates that the current macroblock belongs to the last row of the picture. This field may be used by the kernel to manage pixel output when overlap transform is on. 0 = Not in the last row 1 = In the last row
	3	**LastMBInRow – This field indicates the last MB in row flag.**
	2:0	**Reserved.** MBZ
+1	32:26	**Reserved.** MBZ
	25:24	**OSEdgeMaskChroma** This field contains the overscan edge mask for the Chroma blocks. The left edge masks are hardware and the top edge masks are used by the kernel software. Bit 24: Top edge of block Cb/Cr Bit 25: Left edge of block Cb/Cr
	23:16	**OSEdgeMaskLuma** This field contains the overscan edge mask for the Luma blocks. The left edge masks are hardware and the top edge masks are used by the kernel software. Bit 16: Top edge of block Y0 Bit 17: Top edge of block Y1 Bit 18: Top edge of block Y2

DWord	Bits	Description
		Bit 19: Top edge of block Y3 Bit 20: Left edge of block Y0 Bit 21: Left edge of block Y1 Bit 22: Left edge of block Y2 Bit 23: Left edge of block Y3 *Programming Note: In order to create 8 predication bits from each edge mask bit, software may first create a 0, 1 vector by using a shr instruction with a step shift vector like 0, 1, 2, 3 (e.g. using immediate of type :v. Then each 0 or 1 of the LSB can be repeated by an and instruction to create 8 bits to the flag register. Alternatively, this can be achieved with one and instruction using a CURBE constant map of bit 0 and bit 1 mask.*
	15:8	**VertOrigin - Vertical Origin** In unit of macroblocks relative to the current picture (frame or field).
	7:0	**HorzOrigin - Horizontal Origin** In unit of macroblocks.
+2	31:16	**MotionVector[0].Vert**
	15:0	**MotionVector[0].Horz**
+3	31:0	**MotionVector[1]**
+4	31:0	**MotionVector[2]**
+5	31:0	**MotionVector[3]**
+6	31:0	**MotionVectorChroma** This field is not valid for a field motion in an interlaced frame picture where 4 MVs for chroma blocks. Notes: This field is derived from MotionVector[3:0] as described in the following section.

<table>
<tr><td>+7</td><td>31:24</td><td>Subblock Code for Y3
The following subblock coding definition applies to all 6 subblock coding bytes. Bits 7:6 are reserved.
<table>
<tr><th colspan="2">Subblock Partitioning
(Bits [1:0])
Specify Transform uses for an 8x8 block</th><th colspan="4">Subblock Present
(0 means not present, 1 means present)</th></tr>
<tr><th>Bits [1:0]</th><th>Meaning</th><th>Bit 2</th><th>Bit 3</th><th>Bit 4</th><th>Bit 5</th></tr>
<tr><td>00</td><td>Single 8x8 block (sb0)</td><td>Sb0</td><td>Don’t care</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>01</td><td>Two 8x4 subblocks (sb0-1)</td><td>Sb1 (bot)</td><td>Sb0 (top)</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>10</td><td>Two 4x8 subblocks (sb0-1)</td><td>Sb1 (right)</td><td>Sb0 (left)</td><td>Don’t care</td><td>Don’t care</td></tr>
<tr><td>11</td><td>Four 4x4 subblocks (sb0-3)</td><td>Sb3 (lower right)</td><td>Sb2 (lower left)</td><td>Sb1 (upper right)</td><td>Sb0 (upper left)</td></tr>
</table></td></tr>
<tr><td></td><td>23:16</td><td>Subblock Code for Y2</td></tr>
<tr><td></td><td>15:8</td><td>Subblock Code for Y1</td></tr>
<tr><td></td><td>7:0</td><td>Subblock Code for Y0</td></tr>
<tr><td>+8</td><td>31:16</td><td>Reserved. MBZ</td></tr>
<tr><td></td><td>15:8</td><td>Subblock Code for Cr</td></tr>
<tr><td></td><td>7:0</td><td>Subblock Code for Cb</td></tr>
<tr><td>+9</td><td>31:24</td><td>ILDB control data for block Y3</td></tr>
<tr><td></td><td>23:16</td><td>ILDB control data for block Y2</td></tr>
<tr><td></td><td>15:8</td><td>ILDB control data for block Y1</td></tr>
<tr><td></td><td>7:0</td><td>ILDB control data for block Y0</td></tr>
<tr><td>+10</td><td>31:16</td><td>Reserved</td></tr>
<tr><td></td><td>15:8</td><td>ILDB control data for Cr block</td></tr>
<tr><td></td><td>7:0</td><td>ILDB control data for Cb block</td></tr>
</table>

Indirect Data Format in VC1-IT Mode

VC1-IT mode only contains IT-COEFF indirect data which is described in *Common Indirect IT-COEFF Data Structure*.

Inline Data Description in MPEG2-IT Mode

The content in this command is similar to that in the MEDIA_OBJECT command in IS mode described in the Media Chapter.

Each MFD_IT_OBJECT command corresponds to the processing of one macroblock. Macroblock parameters are passed in as inline data and the non-zero DCT coefficient data for the macroblock is passed in as indirect data.

Inline Data Description in MPEG2-IT Mode depicts the inline data format. Inline data starts at dword 7 of MFD_IT_OBJECT command. There are 7 dwords total.

Inline data in MPEG2-IT Mode

<table>
<tr><th>DWord</th><th>Bit</th><th>Description</th></tr>
<tr><td>+0</td><td>31:28</td><td>Motion Vertical Field Select. A bit-wise representation of a long [2][2] array as defined in §6.3.17.2 of the ISO/IEC 13818-2 (see also §7.6.4).
<table>
<tr><th>Bit</th><th>MVector[r]</th><th>MVector[s]</th><th>MotionVerticalFieldSelect Index</th></tr>
<tr><td>28</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>29</td><td>0</td><td>1</td><td>1</td></tr>
<tr><td>30</td><td>1</td><td>0</td><td>2</td></tr>
<tr><td>31</td><td>1</td><td>1</td><td>3</td></tr>
</table>
Format = MC_MotionVerticalFieldSelect.
0 = The prediction is taken from the top reference field.
1 = The prediction is taken from the bottom reference field.</td></tr>
<tr><td></td><td>27</td><td>Reserved (was Second Field)</td></tr>
<tr><td></td><td>26</td><td>Reserved. (HWMC mode)</td></tr>
<tr><td></td><td>25:24</td><td>Motion Type. When combined with the destination picture type (field or frame) this Motion Type field indicates the type of motion to be applied to the macroblock. See ISO/IEC 13818-2 §6.3.17.1, Tables 6-17, 6-18. In particular, the device supports dual-prime motion prediction (11) in both frame and field picture type.
Format = MC_MotionType
<table>
<tr><th>Value</th><th>Destination = Frame
Picture_Structure = 11</th><th>Destination = Field
Picture_Structure != 11</th></tr>
<tr><td>‘00’</td><td>Reserved</td><td>Reserved</td></tr>
<tr><td>‘01’</td><td>Field</td><td>Field</td></tr>
<tr><td>‘10’</td><td>Frame</td><td>16x8</td></tr>
<tr><td>‘11’</td><td>Dual-Prime</td><td>Dual-Prime</td></tr>
</table></td></tr>
</table>

DWord	Bit	Description
	23:22	Reserved. (Scan method)
	21	**DCT Type.** This field specifies the DCT type of the current macroblock. The kernel should ignore this field when processing Cb/Cr data. See *ISO/IEC 13818-2* §6.3.17.1. This field is zero if Coded Block Pattern is also zero (no coded blocks present). 0 = MC_FRAME_DCT (Macroblock is frame DCT coded). 1 = MC_FIELD_DCT (Macroblock is field DCT coded).
	20	Reserved (was Overlap Transform - H261 Loop Filter).
	19	Reserved (was 4MV Mode - H263)
	18	**Macroblock Motion Backward.** This field specifies if the backward motion vector is active. See *ISO/IEC 13818-2* Tables B-2 through B-4. 0 = No backward motion vector. 1 = Use backward motion vector(s).
	17	**Macroblock Motion Forward.** This field specifies if the forward motion vector is active. See *ISO/IEC 13818-2* Tables B-2 through B-4. 0 = No forward motion vector. 1 = Use forward motion vector(s).
	16	**Macroblock Intra Type.** This field specifies if the current macroblock is intra-coded. When set, Coded Block Pattern is ignored and no prediction is performed (i.e., no motion vectors are used). See *ISO/IEC 13818-2* Tables B-2 through B-4. 0 = Non-intra macroblock. 1 = Intra macroblock.
	15:12	**Reserved : MBZ**
	11:6	**Coded Block Pattern.** This field specifies whether blocks are present or not. Format = 6-bit mask. Bit 11: Y0 Bit 10: Y1 Bit 9: Y2 Bit 8: Y3 Bit 7: Cb4 Bit 6: Cr5
	5:4	Reserved. (Quantization Scale Code)
	3	LastMBInRow – This field indicates the last MB in each row.

DWord	Bit	Description
	2:0	Reserved: MBZ
+1	31:16	**Reserved : MBZ**
	15:8	**VertOrigin - Vertical Origin** In unit of macroblocks relative to the current picture (frame or field).
	7:0	**HorzOrigin - Horizontal Origin** In unit of macroblocks.
+2	31:16	**Motion Vectors – Field 0, Forward, Vertical Component.** Each vector component is a 16-bit two's-complement value. The vector is relative to the current macroblock location. According to ISO/IEC 13818-2 Table 8, the valid range of each vector component is [-2048, +2047.5], implying a format of s11.1. However, it should be noted that motion vector values are sign extended to 16 bits.
	15:0	**Motion Vectors – Field 0, Forward, Horizontal Component**
+3	31:16	**Motion Vectors – Field 0, Backward, Vertical Component**
	15:0	**Motion Vectors – Field 0, Backward, Horizontal Component**
+4	31:16	**Motion Vectors – Field 1, Forward, Vertical Component**
	15:0	**Motion Vectors – Field 1, Forward, Horizontal Component**
+5	31:16	**Motion Vectors – Field 1, Backward, Vertical Component**
	15:0	**Motion Vectors – Field 1, Backward, Horizontal Component**

Indirect Data Format in MPEG2-IT Mode

MPEG2-IT mode only contains IT-COEFF indirect data which is described in Section *Common Indirect IT-COEFF Data Structure*.

MFX Deblocking Commands

Following are MFX Deblocking Commands:

MFX_DBK_OBJECT

MFX Error Handling

This topic is currently under development.

Encoder StreamOut Mode Data Structure Definition

When StreamOut is enabled, per MB (and/or per Slice, per Picture) intermediated coding data (for example, bit allocated for each MB, and so on) are sent to the memory in a fixed record format (and of fixed size) from the PAK. The per-MB records must be written in a strict raster order and with no gap (that is, every MB regardless of its mb_type and slice type, must have an entry in the StreamOut buffer). Therefore, the consumer of the StreamOut data can offset into the StreamOut Buffer (**StreamOut Data Destination Base Address**) using individual MB addresses.

Adding per macroblock stream out for PAK is for the following purposes:

- Immediate multi-pass PAK (without host or EU intervention)
  - 3200-bit conformance
  - Re-quantization
- Providing information for host for offline processing
- Providing information for updated QP's

The description for the fixed format PAK streamout record:

Streamout Pointer: Use the existing streamout pointer and enabler

Per Macroblock Information (a fixed size structure)

DWord	Bit	Description
0	31:24	**MbQpY** - Actual QPY used by the macroblock.
	23:16	**[CHV] MbClock16** – MB compute clocks in 16-clock unit.
	15:8	Reserved: MBZ
	7:4	Reserved: MBZ (future conformance flags)
	3	Reserved
	2	**MbRcFlag:** MB level Rate control flag(pass through) The same value as **RateControlCounterEnable** of MFX_AVC_SLICE_STATE Command
	1	**MbInterConfFlag:** MB level InterMB conformance flag to trigger mutli-pass 1- if total Bit Count of an inter macroblock is more than Inter Conformance Max size limit in the MFX_AVC_IMG_STATE Command
	0	**MbIntraConfFlag:** MB level IntraMB conformance flag to trigger mutli-pass 1- if total Bit Count of an intra macroblock is more than Intra Conformance Max size limit in the MFX_AVC_IMG_STATE Command
1	31:29	Reserved
	28:16	**MbBits:** Total Bit Count for the macroblock
	15:12	Reserved
	12:0	**MbHdrBits:** Header Bit count (bit count due to Pre-coefficient data) for the macroblock
2	31:27	Reserved
	26:0	**Cbp:** Coded Block Pattern of sub-blocks

DWord	Bit	Description
3	31:30	Reserved
	29	**IntraMBFlag**
	28:24	**MBType5Bits**
	23:17	Reserved
	16	**ClampFlag:** Coefficient clamping flag for RC (Status) 1 - Indicates if clamping of any coefficient is done on the macroblock for Rate Control
	15:0	Reserved (future QRC stat output)

PAK Multi-Pass

Multi-Pass PAK Usages:

- Intra MB 3200-bit conformance
- Inter MB Re-quantization
- Frame level Re-quantization

How to Enable Multi-Pass PAK?

- Using the existing conditional batch buffer execution capability to skip/execute the second pass
    - How to dynamically change the condition?
        - Defined one error condition register with a mask. Do HW status page update at the end of the first pass. 0 means all OK, non-zero means there is an error condition, requiring second pass. Mask is used by the host to control what kind of multi-pass is intended.
        - For example, one error bit is 3200-bit conformance violation. Another error bit is the total bit count exceeds (too much or too little) the target range (need to define the target range in the state).
        - **The logic pefectly fits in the conditional batch buffer control logic that VCS has today in GT. There is no additional logic need to be added in VCS to support media functionality.** (**Batch Buffer Skip:** This field only takes effect if Compare Semaphore is set and the value at **Semaphore Address is NOT greater than the Semaphore Data Dword)**.
- Adding a picture level state command to enable and control the behavior of the second pass PAK
    - How to control the re-PAK? Added 3 conformance flags (error registers) in the per-MB streamout. Then the error control is based on the error register and the mask defined in picture level states. There are 8 register flags defined out of which only the 3200-bit case has usage model defined for today. The rest are left for future usage.

Issues and Limitations:

- There is no programmable engine in MFX for flexible control: Therefore, whatever we have defined must consider flexibility

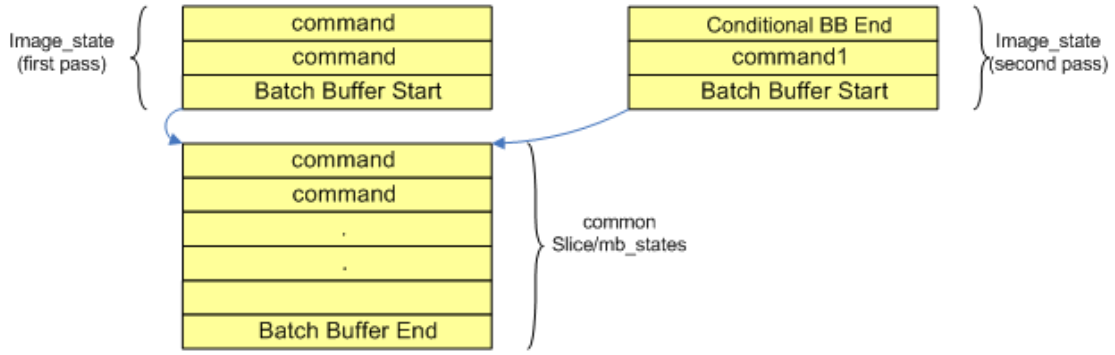
Following 2 MI packets are used inside VCS without any change to support Multipass-PAK behaviour.

- MI_Conditional_Batch_Buffer_End
- Memory Interface Registers

Driver Usage

Driver places Image states in one batch buffer and all slice level and macroblock level states into another batch buffer and link 2 batch buffers. Also replicate Image states with multipass changes in another batch buffer link them to slice/macroblock batch buffer. In this way, only Image states are replicated but not the slice/macroblock states. The image states includes all buffers defined at image(indirectMV, original pixel buffer, etc). Following changes are needed in the Multipass Image State

- **Reset- Stream-Out Enable(disable stream out in the second pass)**
- **Set- MacroblockStatEnable (enable reading of macroblock status buffer)**
- **Reset- 3200-bit conformance (do not report 3200-bit conformance)**



Define Conditional Batch Buffer End for CS/VCSVINunit

Programming Reference

This topic is currently under development.

Monochrome Picture Processing

Monochrome picture is specified using the Surface State with Surface Format of 12. Therefore, MFX hardware, in either decode or encode mode, does not generate any read or write traffic for U/V components. The motivation for this bandwidth optimization is that monochrome video coding might be used for wireless display.

For Encoder:

1. No read in UV original components
2. Processing UV component - no
3. Reconstructed UV component reference picture - no
4. Filter UV component - no

For Decoder:

1. VLD mode: There is no color component coming out of the decoding pipeline in Monochrome mode and so no processing and not writing output.
2. IT mode: There is no color component in the coefficient buffer, and so no processing and not writing output.

Context Switch

There is no pre-emption for the BCS pipeline; hence every command buffer is required to contain all the states setup (preamble). Specifically, CPU can not interrupt the BCS-BSD pipe, to stop the operation in the middle of decoding a bitstream data.

Switch of contexts can only be performed at picture boundary.

No state needs to be saved.

PMSI Support

This topic is currently under development.

Pipeline Flush

Implicit flush for AVC and VC1 is performed at the end of Slice : for MPEG2 is done when a new image/picture command is issued. Because MPEG2 a slice can be one MB, no point to flush. MPEG2 will snoop the next command if it is an img_state command.

Explicit flush MI (1 bit to do media pipeline vs Gx pipeline) flush and cache flush (switch reference frame) – MI flush has bit to do cache flush. MI flush is for driver synchronization.

MMIO Interface

A set of registers are defined and accessible through MMIO interface to serve multiple purposes:

- Use for system configuration
- For accessing Performance counters

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFD ERROR Status	MFD ERROR STATUS_VLD ERROR flags and counter	RO	12400	Dec
Reserved	MBZ		12404~1241C	
MFD picture-level parameter	VC1 picture level parameters	R/W	12420	Dec
Reserved	MBZ		12434	
MFX PIPELINE_STATUS_FLAGS	MFX PIPELINE STATUS Flags_MFX pipeline mode flags	RO	12438	Dec
MFX_Error_Injection_Parameter	Control HW error injector	WO	12454	Dec
Reserved			12458~1245C	
MFX Frame Performance count	Number of clocks spent decoding/encoding a frame	RO	12460	Dec/Enc
MFX Slice Performance count	Number of clocks spent decoding/encoding a slice	RO	12464	Dec/Enc
MFX Frame Macroblock count	Number of MBs decoded/encoded per frame	RO	12468	Dec/Enc
MFD Frame BITSTREAM SE/BIN count	Number of bin/SE decoded per frame	RO	1246C	Dec
MFX Memory Latency count1	Reference picture read latency - min and max	RO	12470	Dec/Enc
MFX Memory Latency count2	Reference picture read latency - Accumulative (used for compute AVE latency)	RO	12474	Dec/ENc
MFX Memory Latency count3	row-store/bit-stream memory read latency -min and max	RO	12478	Dec/Enc
MFX Memory Latency count4	row-store/bit-stream memory read latency - accumulative (used to compute AVE latency)	RO	1247C	Dec/End
MFX Frame row-stored/bit-stream read Count	# of row-store memory requests sent	RO	12480	Dec/End
MFX Motion Comp read Count	total number of CL memory accesses per frame	RO	12484	Dec/ENd
MFX Motion Comp MISS Count	total number of CL HITs per frame	RO	12488	Dec/ENd
Reserved			1248C~1249C	

MFC_BITSTREAM_BYTECOUNT_FRAME	Total Bitstream Output Byte Count register per Frame	RO	124A0	Enc
MFC_BITSTREAM_SE_BITCOUNT_FRAME	Bitstream Output total Byte Count for syntax eements (total byes of MB data from SEC per frame)	RO	124A4	Enc
MFC_AVC_CABAC_BIN_COUNT_FRAME	Bitstream Output total bin count per frame	RO	124A8	Enc
MFC_AVC_CABAC_INSERTION_COUNT	Bitstream Output CABAC Insertion Count Register	RO	124AC	Enc
MFC_AVC_MINSIZE_PADDING_COUNT	Bitstream Output Minimal Size Padding Count Register	RO	124B0	Enc
MFC_IMAGE_STATUS_MASK	image status(flags).	R/W	124B4	Enc
MFC_IMAGE_STATUS_CONTROL	suggested data for next frame in multi-pass.	RO	124B8	Enc
MFC_QP_STATUS_COUNT	Overall adjusted delta QP via multi-pass, Sum of QPY for all macroblocks of the frame	RO	124BC	Enc
			124C0~124CC	Enc
MFC_BITSTREAM_BYTECOUNT_SLICE	Bitstream Output Byte Count Register per Slice	RO	124D0	Enc
MFC_BITSTREAM_SE_BITCOUNT_SLICE	Bitstream Output Bit Count for the last Syntax Element Register	RO	124D4	Enc
PAK_ REPORT_WARNING	MPC Warning Register	RO	124E4	Enc
PAK_REPORT_ERROR	MPC Error Register	RO	124E8	Enc
PAK_REPORT_RUNNING	PAK_REPORT_RUNNING status register	RO	124EC	Enc
Reserved			124F0~124FC	Enc

Decoder Registers

Following are Decoder Registers:

MFD_ERROR_STATUS - MFD Error Status

AVC CAVLC

AVC CABAC

VC1

MPEG2

JPEG

MFD_PICTURE_PARAM - MFD Picture Parameter

MFX_STATUS_FLAGS - MFX Pipeline Status Flags

MFX_FRAME_PERFORMANCE_CT - MFX Frame Performance Count

MFX_SLICE_PERFORM_CT - MFX Slice Performance Count

MFX_MB_COUNT - MFX Frame Macroblock Count

MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count

MFX_LAT_CT1 - MFX_Memory_Latency_Count1

MFX_LAT_CT2 - MFX Memory Latency Count2

MFX_LAT_CT3 - MFX Memory Latency Count3

MFX_LAT_CT4 - MFX Memory Latency Count4

MFX_SE-BIN_CT - MFX Frame BitStream SE/BIN Count

MFX_READ_CT - MFX Frame Motion Comp Read Count

MFX_MISS_CT - MFX Frame Motion Comp Miss Count

Encoder Registers

Following are the Encoder Registers:

MFC_VIN_AVD_ERROR_CNTR - MFC_AVC Bitstream Decoding Front-End Parsing Logic Error Counter

MFC_BITSTREAM_BYTECOUNT_FRAME - Reported Bitstream Output Byte Count per Frame Register

MFC_BITSTREAM_SE_BITCOUNT_FRAME - Reported Bitstream Output Bit Count for Syntax Elements Only Register

MFC_AVC_CABAC_BIN_COUNT_FRAME - Reported Bitstream Output CABAC Bin Count Register

AVC_CABAC_INSERTION_COUNT - MFC_AVC_CABAC_INSERTION_COUNT

MFC_AVC_MINSIZE_PADDING_COUNT - Bitstream Output Minimal Size Padding Count Report Register

MFC_IMAGE_STATUS_MASK - MFC Image Status Mask

MFC_IMAGE_STATUS_CONTROL - MFC Image Status Control

MFC_QUP_CT - MFC QP Status Count

MFC_BITSTREAM_BYTECOUNT_SLICE - Bitstream Output Byte Count Per Slice Report Register

MFC_BITSTREAM_SE_BITCOUNT_SLICE - Bitstream Output Bit Count for the last Syntax Element Report Register

MFX_PAK_ERROR Register

MFX_PAK_WARNING Register

MFX_VP8_CNTRL_MASK - Reported BitRateControl parameter Mask

MFX_VP8_CNTRL_STATUS - Reported BitRateControl parameter Status

MFX_VP8_FRM_BYTE_CNT - Reported Final Bitstream Byte Count

MFX_VP8_FRM_ZERO_PAD - Reported Frame Zero Padding Byte Count

MFX_VP8_BRC_DQindex - Reported BitRateControl DeltaQindex

MFX_VP8_BRC_DLoopFilter - Reported BitRateControl DeltaLoopFilter

MFX_VP8_BRC_CumulativeDQindex01 - Reported BitRateControl CumulativeDeltaQindex and Qindex 01

MFX_VP8_BRC_CumulativeDQindex23 - Reported BitRateControl CumulativeDeltaQindex and Qindex 23

MFX_VP8_BRC_CumulativeDLoopFilter01 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 01

MFX_VP8_BRC_CumulativeDLoopFilter23 - Reported BitRateControl CumulativeDeltaLoopFilter and LoopFilter 23

MFX_VP8_BRC_Convergence_Status - Reported BitRateControl Convergence Status

MMIO Interface

A set of registers are defined and accessible through MMIO interface to serve multiple purposes:

- Use as Status register for Bit Rate Control
- Use for Context Switch in Multipass

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	12900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	12904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	12908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1290B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	12910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	12914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW (RO for CHV A0)	12918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW (RO for CHV A0)	1291C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW (RO for CHV A0)	12920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW (RO for CHV A0)	12924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW (RO for CHV A0)	12928	Enc

The following registers are the same as above except they have a different Address Offset. They are used if the second VDbox (VP8 Encoder) exists.

Register Name	Description	Register Type	Address Offset	Dec/Enc
MFX_VP8_CNTRL_MASK	BitRateControl parameter Mask register	RO	1C900	Enc
MFX_VP8_CNTRL_STATUS	BitRateControl parameter Status register	RO	1C904	Enc
MFX_VP8_FRM_BYTE_CNT	Final Bitstream Byte count	RO	1C908	Enc
MFX_VP8_FRM_ZERO_PAD	Final Bitstream Zero Padding Byte count	RO	1C90B	Enc
MFX_VP8_BRC_DQindex	BitRateControl Delta Qindex	RO	1C910	Enc
MFX_VP8_BRC_DLoopFilter	BitRateControl Delta LoopFilter	RO	1C914	Enc
MFX_VP8_BRC_CumulativeDQindex01	BitRateControl Cumulative Delta Qindex for Seg0/1	RW (RO for CHV A0)	1C918	Enc
MFX_VP8_BRC_CumulativeDQindex23	BitRateControl Cumulative Delta Qindex for Seg2/3	RW (RO for CHV A0)	1C91C	Enc
MFX_VP8_BRC_CumulativeDLoopFilter01	BitRateControl Cumulative Delta LoopFilter for Seg0/1	RW (RO for CHV A0)	1C920	Enc
MFX_VP8_BRC_CumulativeDLoopFilter23	BitRateControl Cumulative Delta LoopFilter for Seg2/3	RW (RO for CHV A0)	1C924	Enc
MFX_VP8_BRC_Convergence_Status	BitRateControl Convergence Status	RW (RO for CHV A0)	1C928	Enc

Row Store Sizes and Allocations

	AVC	VC1	MPEG2	JPEG	IT	ENC	SEC ENC
vin_vmx_pixcoefind_ addr[31:6]	Bitstream	Bitstream	Bitstream	Bitstream	VDS COEF	Orig Pix	BSP data
vin_vmx_mvbsdrs_ addr[31:6]	VAD BSD		VMD RS		VDS MV	MPC MV	
vin_vmx_mpcildbmpr_ addr[31:6]	VAM MPR				VDS ILDB	MPC RS	
vin_vmx_dmv*_ addr[31:6]	VAM DMV	VCP DMV					
vin_vmx_bp_addr [31:0]		VCP BP					

MPEG2 VLD Decoding Mode :

use BSD Row Store only, and

MPEG2 IT Decoding Mode :

MPEG2 IT mode does not need row-store

JPEG VLD Decoding Mode : no row store is needed

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 9: Media VEBOX

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Media VEBOX Introduction 1

Denoise 1

Motion Detection and Noise History Update 2

Temporal Filter 2

Context Adaptive Spatial Filter 2

Denoise Blend 2

Chroma Noise Reduction 2

Chroma Noise Detection 2

Chroma Noise Reduction Filter 2

Block Noise Estimate (Part of Global Noise Estimate) 3

Deinterlacer 3

Deinterlacer Algorithm 4

Film Mode Detector 4

Image Enhancement Color Processing (IECP) 5

Skin Tone Detection Enhancement (STDE) 5

STD Score Output 5

Adaptive Contrast Enhancement (ACE) 6

Total Color Control (TCC) 6

ProcAmp 7

Color Space Conversion 8

Color Gamut Compression 9

Overview 9

Usage Models 10

Color Correction (Gamut Expansion) 10

Overview 11

Usage Models 11

VEBOX Output Statistics 12

Overall Surface Format 12

Statistics Offsets 13

Per Command Statistics 13

FMD Variances and GNE Statistics 14

Skin-Tone Data 14

Gamut Compression: Out of Range Pixels 15
Histograms 15
Ace Histogram 15
STMM / Denoise 16
VEBOX State and Primitive Commands 17
VEBOX State 17
DN-DI State Table Contents 17
VEBOX_IECP_STATE 18
VEBOX Surface State 20
Surface Format Restrictions 20
VEB DI IECP Commands 21
Command Stream Backend - Video 21
Video Enhancement Engine Functions 21

Media VEBOX Introduction

The VEBOX is an independent pipe with a variety of image enhancement functions.

The following sections are contained in Media VEBOX:

- Denoise
- Deinterlacer
- Image Enhancement/Color Processing (IECP)
- VEBOX Output Statistics
- VEBOX State
- VEBOX Surface State
- VEB DI IECP Commands
- Command Stream Backend - Video
- Video Enhancement Engine Functions

Denoise

This section discusses the Denoise feature in the chipset.

- Denoise Filter - detects noise in the input image and filters the image with either a temporal filter or a spatial filter. The temporal filter is applied when low motion is detected.
- Chroma Denoise Filter - detects noise in the U and V planes separately and applies a temporal filter.
- Block Noise Estimate (BNE) - as part of the Global Noise Estimate (GNE) algorithm, BNE estimates the noise over each block of pixels in the input picture.
- Global Noise Estimate (GNE) - GNE is calculated at the end of the frame by combining all the BNEs. The final GNE value is used to control the denoise filter for the next input frame. Noise estimates are kept between frames and blended together.

Filters and Functions:

- Temporal filter
- Context Adaptive Spatial Filter
- Denoise Blend
- Chroma Noise Reduction
- Block Noise Estimate

Motion Detection and Noise History Update

This logic detects motion of the current block for the denoise filter, which it then combines with motion detected in the co-located block of the past frame to be stored in the denoise history table. Denoise history is saved to memory and also used to control the temporal denoise filter.

Temporal Filter

Temporal denoise is applied to each pixel based on the noise strength measured from the input pictures. Each pair of co-located pixels in the current and previous input pictures is blended together to generated the output pixel.

Context Adaptive Spatial Filter

For each pixel in the local neighborhood, its luma value is compared (via absolution difference) to the center pixel to be filtered.

Each pixel in the neighborhood for which the absolute difference is less than *good_neighbor_th* is marked as a "good neighbor".

The filtered output pixel is then equal to average of good neighbor pixels.

Denoise Blend

The denoise blend combines the temporal and spatial denoise outputs.

Chroma Noise Reduction

This chapter contains descriptions of filters that support the chroma noise reduction feature in the chipset.

Filters:

- Chroma noise detection
- Chroma Noise Reduction filter

Chroma Noise Detection

The operation of chroma noise detection module is similar to the luma noise detection module, BNE and GNE.

The U & V channels are processed individually to generate a noise estimate for each of the 2 channels.

Chroma Noise Reduction Filter

A simple and effective temporal-domain chroma noise reduction filter is introduced.

The Noise History is updated based on the motion detection result and is saved to the memory.

The Noise History value is used to control the temporal denoise filter.

The output of the temporal filter is computed as a weighted blending of two co-located chroma pixels in the current and previous input pictures.

The Noise History computed between the current and previous input pictures is used to control the strength of this temporal blending of two co-located chroma pixels.

The output of the temporal filter is blended again with the input pixel value in the current picture depending on the motion information.

Block Noise Estimate (Part of Global Noise Estimate)

The BNE estimates the amount of noise in each rectangular region of the input picture. The BNE estimate is computed separately for each color component in the input picture. The estimates from BNE are summed together to generate the Global Noise Estimate (GNE) for the entire input picture.

Deinterlacer

The deinterlacer (DI) takes the top and bottom fields of each input frame and converts them into two individual output frames. This block also gathers statistics for a film mode detector (FMD) that runs in software at the end of the frame. If the film mode detector determines that the input is progressive rather than interlaced, then the input fields are put together to construct the progressive output frame.

Features:

- **Deinterlacer** - estimates how much motion is present across the input fields. Low motion scenes are reconstructed by averaging pixels from temporally nearby fields (temporal deinterlacer), while high motion scenes are reconstructed by interpolating pixels from spatially nearby fields (spatial deinterlacer).
- **Film Mode Detection (FMD)** - determines if the input fields are created by sampling film content and converting it to interlaced video. If so, the deinterlacer is turned off in favor of reconstructing the progressive output frame directly from adjacent fields. Various sum-of-absolute differences are computed per block. The FMD algorithm consumes these variances from all blocks of both input fields at the end of the frame.
- **Progressive Cadence Reconstruction** - If the FMD for the previous input frame determines that film content has been converted into interlaced video, then this block reconstructs the original frame by directly putting together adjacent fields.
- **Chroma Upsampling** - If the input is 4:2:0 then chroma data is doubled vertically to convert it to 4:2:2. Chroma data is then processed by its own version of the deinterlacer or progressive cadence reconstruction algorithms.

Deinterlacer Algorithm

The overall goal of the motion adaptive deinterlacer is to convert an interlaced video stream made of fields of alternating lines into a progressive video stream made of frames in which every line is provided.

If there is no motion in a scene, then the missing lines in the current field picture can be provided by looking at the previous or next field pictures (temporal deinterlacing or TDI). If there is motion in the scene, then objects in the previous and next fields are displaced from the location in the current field, so motion estimation and compensation are required to deinterlace using the temporally neighboring field pictures. Instead, spatial interpolation from the neighboring lines above and below in the current field picture is used to fill in the missing lines (spatial deinterlacing or SDI).

The motion adaptive deinterlacing is implemented by computing a measure of motion called the Spatial-Temporal Motion Measure (STMM). If this measure shows that there is little motion in an area around the current pixel, then the missing pixels/lines are filled in by averaging the pixel values from the previous and next fields. If the STMM shows that there is motion, then the missing pixels/lines are filled in by interpolating from spatially neighboring lines. The two results from TDI and SDI are alpha-blended for intermediate values of STMM to prevent sudden transitions between TDI and SDI modes.

The deinterlacer uses two frames for reference. The current frame contains the field that is being deinterlaced. The reference frame is the closest frame in time to the field that is being deinterlaced. for example, when the 1st field is being deinterlaced, the previous frame is the reference and when the 2nd field is being deinterlaced, it is the next frame that is the reference.

Film Mode Detector

The File Mode Detector (FMD) detects film contents that have been converted to interlaced video in which case each pair of input fields are merged together to a frame picture.

Image Enhancement Color Processing (IECP)

The IECP consists of these functions:

- STD - Skin Tone Detection detects colors that might represent skin.
- STE - Skin Tone Enhancement modifies colors marked by STD.
- GCC - Gamut Compression
- ACE - Automatic Contrast Enhancement changes luma values to enhance contrast.
- TCC - Total Color Control allows UV values to be modified to adjust color saturation.
- ProcAmp - implements the ProcAmp DDI functions to modify the brightness, contrast, hue, and saturation.
- CSC - Color Space Conversion
- GEE - Gamut Expansion and Color Correction in Linear RGB Space

Skin Tone Detection Enhancement (STDE)

The STD/E unit, composed of the Skin Tone Detection (STD) and Skin Tone Enhancement (STE) units, is part of color processing pipe located at the Render Cache Pixel Backend (RCBP).

The main goal of the STD/E is to reproduce the skin colors in a way that is more palatable to the observer, and by that to increase the sensed image quality. It may also pass indication of skin tones to the TCC and ACE.

The STD unit detects the skin-like colors and passes a likelihood score for each input pixel indicating the probability that it is a skin tone color to the STE. The STE modifies the saturation and hue of the pixel according to its skin tone likelihood score. Both the STD and STE evaluations are done on a per-pixel basis. The input pixels are required to be in the YUV space.

The skin tone detection score (that is, skin tone likelihood score) is recorded as a 5-bit number and it is passed to ACE and TCC blocks to indicate the strength of skin tone likelihood.

STD Score Output

When the state bit "Output STD Decisions" is set, STD scores fill the output instead of the pixel values. To output STD scores, STD should be enabled and all other functions in the IECP after STDE, except ACE, should be disabled - only ACE can be enabled to collect the histogram of the STD score values.

The output YUV data when "Output STD Decision" is enabled should be as follows:

- Y = 0x7FF + (STD_Score «6)
- U = 0x7FF
- V = 0x7FF

In this mode, a histogram of skin tone distribution can be obtained in ACE, and a special ACE PWLF curve (step function) can be configured to produce a binary picture to illustrate the pixels based on the level of skin tone detection.

Adaptive Contrast Enhancement (ACE)

Automatic Contrast Enhancement (ACE) is a part of the color processing pipe which is located at the render cache in the RCPB block.

The main goals of ACE are to improve the overall contrast of the image and to emphasize details in obscured regions, such as dark regions of the input image.

The ACE algorithm analyzes the input image and modifies contrast of the image according to its content characteristic. Analysis and contrast adjustment are performed over the Y component.

Total Color Control (TCC)

TCC adjusts the color saturation level of the input image based on six anchor colors (Red, Green, Blue, Magenta, Yellow, and Cyan). The TCC algorithm operates on the UV-color components in the YUV color space on a per-pixel basis.

The input to the TCC block is:

- U and V color components (10 bit)
- Skin-tone detection value (5 bit)
- External control parameters

The output of the TCC block is:

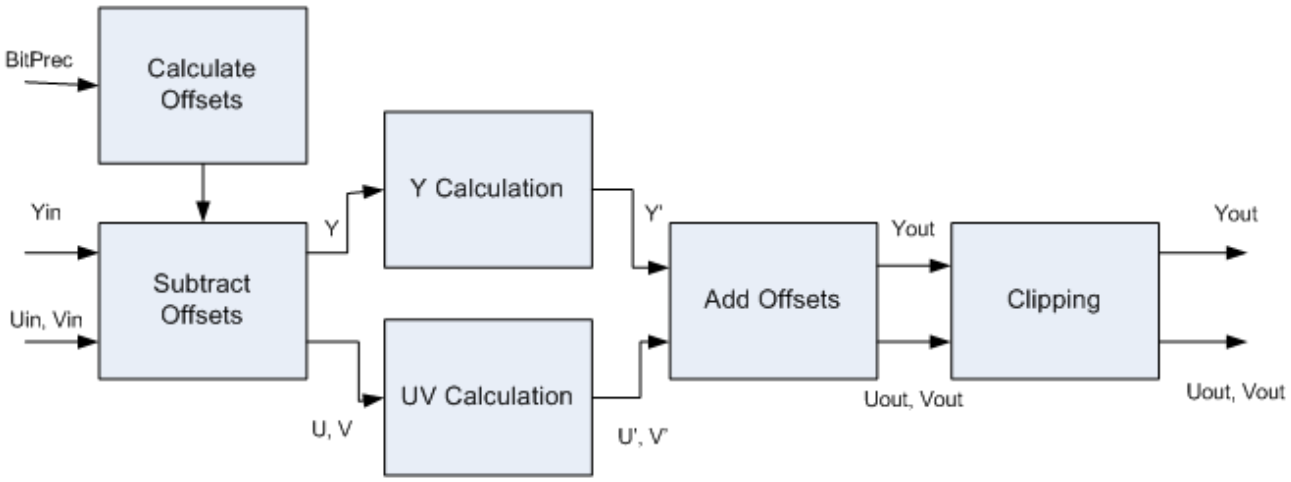
- Updated U and V values (10 bit)

The TCC block is implemented in HW to reduce the power of the system and improve the battery life.

The TCC block is controlled by state only and does not require any memory access. The TCC block runs at the same frequency as the existing RCPBunit.

ProcAmp

The PROCAMP block modifies the brightness, contrast, hue and saturation of the input image in YUV color space.



Y Processing:

An offset of 256 (that is, 16 in 8bpc) is subtracted from the 12-bit Y values to position the black level at zero. This removes the DC offset so that adjusting the contrast does not vary the black level. Since Y values may be less than 256, negative Y values should be supported at this point. Contrast is adjusted by multiplying the YUV pixel values by a constant. If U and V are adjusted, a color shift will result whenever the contrast is changed. The brightness property value is added (or subtracted) from the contrast adjusted Y values; this is done to avoid introducing a DC offset due to adjusting the contrast. Finally the offset 256 is added back to reposition the black level at 256.

The equation for processing of Y values is:

$$Yout' = ((Yin-256) \times C) + B + 256,$$

where C is the Contrast adjustment value and B is the Brightness adjustment value.

UV Processing:

An offset of 2048 (that is, 128 in 8bpc) is subtracted from the 12-bit U and V values. The hue adjustment is implemented by combining the U and V input values together as in:

$$Uout' = (Uin-2048) \times Cos(H) + (Vin-2048) \times Sin(H)$$
$$Vout' = (Vin-2048) \times Cos(H) - (Uin-2048) \times Sin(H)$$

where H represents the desired Hue angle; Saturation is adjusted by multiplying the U and V input values by a constant S.

Finally, the offset value 2048 is added back to both U and V.

The combined processing of Hue, Saturation and Contrast on the UV data is:

```
Uout' = (((Uin-2048) x Cos(H) + (Vin-2048) x Sin(H)) x C x S) + 2048
Vout' = (((Vin-2048) x Cos(H) - (Uin-2048) x Sin(H)) x C x S) + 2048
```

where C is the contrast, H is Hue angle and S is the Saturation.

The multiplication factors Cos(H)x*Cx*S and Sin(H)x*Cx*S are programmed by the parameters Cos_c_s and Sin_c_s.

Color Space Conversion

The CSC block enables linear conversion between different color spaces such as YCbCr and RGB using vector shifts and matrix multiplication.

The CSC algorithm is a linear coordinate transformation, comprising of the following steps:

1. Shift the input color coordinate
2. Multiply by 3x3 matrix
3. Shift the output color coordinate

The formula representation of the 3 steps is:

$$\begin{pmatrix} \text{vout\_1} \\ \text{vout\_2} \\ \text{vout\_3} \end{pmatrix} = \begin{pmatrix} a11 & a12 & a13 \\ a21 & a22 & a23 \\ a31 & a32 & a33 \end{pmatrix} * \begin{pmatrix} \text{vin\_1+v0\_1} \\ \text{vin\_2+v0\_2} \\ \text{vin\_3+v0\_3} \end{pmatrix} + \begin{pmatrix} \text{u0\_1} \\ \text{u0\_2} \\ \text{u0\_3} \end{pmatrix}$$

The output pixel values are clipped to ensure that each color component is within the valid range.

Color Gamut Compression

With the rapid development of capture and display devices, images can be captured, manipulated, and reproduced in a variety of forms. Due to the fact that different devices support different color gamuts, gamut mapping becomes an important feature in media processing where video/image data on multiple platforms are often shared and exchanged. The problems of gamut mapping can be divided into two categories: (1) mapping from wider gamut to narrow gamut, and (2) mapping from narrow gamut to wider gamut. The first category is defined as Color Gamut Compression while the second one is defined as Color Gamut Expansion.

Color Gamut Compression module provides the functionality of mapping the color content in a color gamut wider than that of the output display to the color gamut of the output display while maintaining the hue of the input content. Color Gamut Compression module, for example, maps xvYCC color space to sRGB color space.

The simplest gamut compression method is to clip the out-of-range color values to the valid range (i.e., 0-1 in normalized, linear space). Although this simple clipping method leads to acceptable visual appearance in some cases, the loss of color depth can be problematic as multiple out-of-range color values are mapped to the same color at the gamut boundary. This simple clipping method treats each color channel (i.e., R/G/B) independently and this may also lead to unexpected color distortion since the composite ratio of three primaries (i.e., color hue) is changed.

An advanced approach takes these two factors into account, maintains the original color content information of the image after gamut compression, and is capable of producing output pictures that are more visually pleasant than those produced by the simple clipping.

Overview

The main goal of color gamut compression algorithm is to compress out-of-range pixel values while keeping their hue values same as before.

At the IECP pipeline level, the input to the gamut compression unit comes from the STDE unit and the output of the gamut compression goes to the TCC unit.

The gamut compression comprises of the following stages:

- xvYCC decoding
- YUV2LCH color space conversion
- Fixed-hue Gamut Compression
- xvYCC encoding

Usage Models

There are two usage models of the gamut compression module:

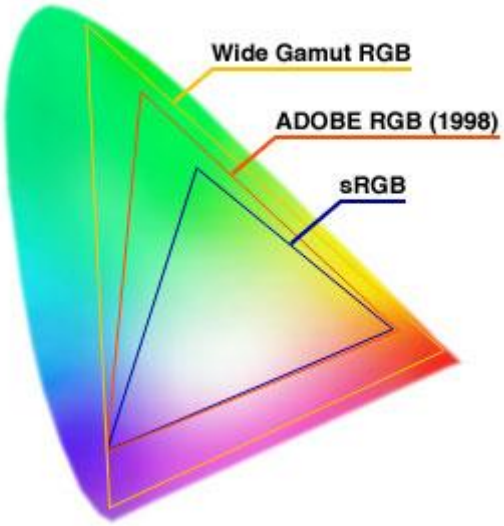
- Basic mode: fixed-hue color gamut clipping mode
- Advanced mode: fixed-hue full range mapping mode

The application of basic mode (that is, fixed-hue color gamut clipping) is preferred when the content has a smaller percentage of out-of-range pixels in the scene. The advanced mode (that is, fixed-hue full range mapping) may change the color of the in-range pixels in addition to the color of the out-of-range pixels and is thus preferred when the percentage of out-of-range pixels is high. The percentage of the out-of-range pixels is derived from the out-of range color gamut detection module to provide an indicator to select between basic mode and advanced mode.

Color Correction (Gamut Expansion)

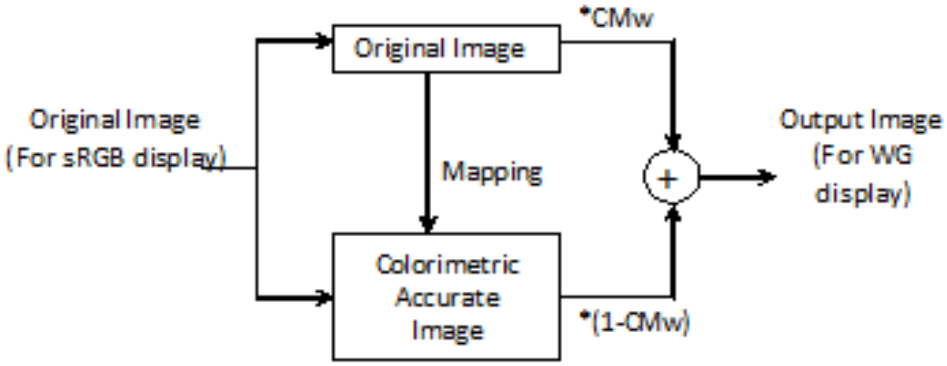
Color Correction is an important and commonly used feature where input RGB colors are modified to output RGB colors in linear RGB space. Color Correction shares the same HW with Gamut Expansion and all descriptions of the Gamut Expansion process in this section apply equally to the Color Correction usage of the HW.

An increasing number of wide gamut (WG) displays are available, which provide additional colors over the traditional display. While most photography today complies with the sRGB standard color space, which covers around 72% of the color perceived by humans, this sRGB color content looks incorrect/unnatural on wide gamut displays. Therefore, a gamut mapping (GM) algorithm is required to adjust the input gamut range to fit the output gamut range.



Overview

The main goal of the gamut expansion algorithm is to produce an output image as the composite of the original image and the accurate-color-reproduction image as shown below:



Gamut Expansion: The image output for a WG display is composed of the original image and the colorimetric accurate image.

Usage Models

There are two usage models:

- Basic mode: global color gamut expansion mode
- Advanced mode: pixel adaptive color gamut expansion mode

The basic mode (global color gamut expansion) provides uniform blending among the colorimetric accurate color and the original color stretched from the color primaries of the wider color gamut display.

The advanced mode (pixel adaptive color gamut expansion) provides per-pixel adaptive weighting to take advantage of the property of the extended color gamut of the current display panel. The pixel adaptive color gamut expansion mode is based on the characteristics of the currently available wide gamut panels. The global color gamut expansion mode may fit in the usage model if the property of the future wide gamut display panels allows it. This is subject to the application configuration upon the product delivery.

VEBOX Output Statistics

Overall Surface Format

Statistics are gathered on both per-block (16x4) basis and per-frame basis. There are 16 bytes of encoder statistics data per 16x4 block, plus a variety of per frame data which are stored in a linear surface. The 16 bytes of encoder statistics per block are output if either DN or DI are enabled and are organized into a surface with a pitch equal to the output surface width rounded to the next higher 64 boundary (so that each line starts and ends on a cache line boundary). The height of the surface is ¼ of the height of the output surface. If both DN and DI are disabled then the encoder stats are not output and the per frame information is output at the base address.

The per frame information is written twice per frame to allow for a 2 slice solution - in a single slice the second set of data will be all 0. The final per frame information is found by adding each individual Dword, clamping the data (except for the ACE histogram, which is 24-bits in each 32-bit Dword) to prevent it from overflowing the Dword.

The Deinterlacer outputs two frames for each input frame. For the case of DN and no DI, only the first set of per frame statistics will be written.

Statistics Surface when DI Enabled and DN either On or Off

16 bytes for X=0, Y=0	16 bytes for X=16, Y=0	16 bytes for X=32, Y=0	• • •
16 bytes for X=0, Y=4			
16 bytes for X=0, Y=8			
•			
•			
•			
16 bytes for X=0, Y=height-4			

256 Dwords of ACE histogram for Slice 0 (Previous Frame)	17 DW Reserved		2 DW of STD 0	2 DW of GCC 0	11 DW of Reserved
256 Dwords of ACE histogram for Slice 0 (Current Frame)	11 DW of FMD 0	6 DW of GNE 0	2 DW of STD 0	2 DW of GCC 0	11 DW of Reserved
256 Dwords of ACE histogram for Slice 1 (Previous Frame)	17 DW Reserved		2 DW of STD 1	2 DW of GCC 1	11 DW of Reserved
256 Dwords of ACE histogram for Slice 1 (Current Frame)	11 DW of FMD 1	6 DW of GNE 1	2 DW of STD 1	2 DW of GCC 1	11 DW of Reserved

Statistics Surface when DN Enabled and DI Disabled

16 bytes for X=0, Y=0	16 bytes for X=16, Y=0	16 bytes for X=32, Y=0	• • •
16 bytes for X=0, Y=4			
16 bytes for X=0, Y=8			
•			
•			
•			
16 bytes for X=0, Y=height-4			

256 Dwords of ACE histogram for Slice 0 (Input Frame)	11 DW of FMD 0	6 DW of GNE 0	2 DW of STD 0	2 DW of GCC 0	11 DW of Reserved
256 Dwords of ACE histogram for Slice 1 (Input Frame)	11 DW of FMD 1	6 DW of GNE 1	2 DW of STD 1	2 DW of GCC 1	11 DW of Reserved

Statistics Surface when both DN and DI Disabled

256 Dwords of ACE histogram for Slice 0	17 DW Reserved	2 DW of STD 0	2 DW of GCC 0	11 DW of Reserved
256 Dwords of ACE histogram for Slice 1	17 DW Reserved	2 DW of STD 1	2 DW of GCC 1	

When DN and DI are both disabled, only the per frame statistics are written to the output at the base address.

Statistics Offsets

The statistics have different offsets from the base address depending on what is enabled.

The encoder statistics size is based on the frame size:

```
Encoder_size = width * (height+3)/4
```

where

width is the width of the output surface rounded to the next higher 64 boundary and

height is the output surface height in pixels.

Offset	DI on	DI off + DN on	DI off + DN off
ACE_Histo_Previous_Slice0	Encoder_size	N/A	N/A
Per_Command_Previous_Slice0	Encoder_size + 0x400	N/A	N/A
ACE _Histo_Current_Slice0	Encoder_size + 0x480	Encoder_size	0x0
Per_Command_Current_Slice0	Encoder_size + 0x880	Encoder_size + 0x400	0x400
ACE_Histo_Previous1	Encoder_size + 0x900	N/A	N/A
Per_Command_Previous_Slice1	Encoder_size + 0xD00	N/A	N/A
ACE_Histo_Current1	Encoder_size + 0xD80	Encoder_size + 0x480	0x480
Per_Command_Current_Slice1	Encoder_size + 0x1180	Encoder_size + 0x880	0x880

Per Command Statistics

The Per Command Statistics are placed after the encoder statistics if either DN or DI is enabled. If the frame is split into multiple calls to the VEBOX, each call outputs only the statistics gathered during that call and software provides different base address per call and sums the resulting output to compute the per-frame data.

The final address of each statistic is:

Statistics Output Address + Per_Command_Offset (pick the one for the slice desired and the current/previous frame for Deinterlacer) **+ PerStatOffset**

FMD Variances and GNE Statistics

These are the 11 FMD variances (Variance 0 ~ 10) and Global Noise Estimate Statistics (Sums and Counts) collected in each VEBOX call.

Note that pixel values in blocks that are close to the edge of the frame (within a 16x4 block that intersects or touches the frame edge) are not used in the variance computation.

FMD variances are 0 when the Deinterlacer is disabled.

GNE estimates are 0 when the Denoise is disabled.

Counter Id	PerStatOffset	Associated Counter
0	0x00	FMD Variance 0
1	0x04	FMD Variance 1
2	0x08	FMD Variance 2
3	0x0C	FMD Variance 3
4	0x10	FMD Variance 4
5	0x14	FMD Variance 5
6	0x18	FMD Variance 6
7	0x1C	FMD Variance 7
8	0x20	FMD Variance 8
9	0x24	FMD Variance 9
10	0x28	FMD Variance 10
11	0x2C	GNE Sum Luma (Sum of BNEs for all passing blocks)
12	0x30	GNE Sum Chroma U
13	0x34	GNE Sum Chroma V
14	0x38	GNE Count Luma (Count of number of block in GNE sum)
15	0x3C	GNE Count Chroma U
16	0x40	GNE Count Chroma V

Skin-Tone Data

The register Ymax stores the largest luma value. It is reset at the start of a command to zero.

The register Ymin stores the smallest luma value. It is reset at the start of a command to:

Reset Value
0x3FF (1023 in 10 bits)

There is also a 29-bit counter of all the skin pixels (Number of Skin Pixles).

Register values are 0 if the STD/STE function is disabled.

The registers are stored with the offsets as shown below.

PerStatOffset	Associated Register
0x044	Ymax (bits 25:16), Ymin (bits[9:0]), other bits zero
0x048	Number of Skin Pixels (bits [28:0], other bits zero)

Gamut Compression: Out of Range Pixels

The statistics gathered for Gamut Compression are:

1. Count of out-of-range pixels (29 bits) and
2. Sum of the distances from out-of-range pixels to the closest range boundaries (32 bits).

If the sum is greater than the maximum 32-bit value, then it is clamped to the maximum 0xFFFFFFFF.

Both values are reset to zero at the start of each command.

Both values are zero if the GCC function is disabled.

PerStatOffset	Associated Register
0x04C	Sum of distances of out-of-range pixels (clamped to 0xFFFFFFFF)
0x050	Number of out-of-range pixels (bits [28:0], other bits are zero)

Histograms

The histograms are included in the main statistics surface along with the encoder statistics and other per command statistics.

Ace Histogram

The Ace Histogram counts the number of pixels at different luma values.

It has 256 bins, each of which is 24 bits.

Any count that exceeds 24 bits is clamped to the maximum value.

The data is stored on DWord boundaries with the upper 8 bits equal to zero.

Y[9:2]	PerStatOffset	Associated Counter
0	0x000	ACE histogram, bin 0
1	0x004	ACE histogram, bin 1
2	0x008	ACE histogram, bin 2
...	...	...
255	0x3fc	ACE histogram, bin 255

STMM / Denoise

The STMM/Denoise history is a custom surface used for both input and output. The previous frame information is read in for the DN (Denoise History) and DI (STMM) algorithms; while the current frame information is written out for the next frame.

STMM / Denoise Motion History Cache Line

Byte	Data
0	STMM for 2 luma values at luma Y=0, X=0 to 1
1	STMM for 2 luma values at luma Y=0, X=2 to 3
2	Luma Denoise History for 4x4 at 0,0
3	Not Used
4-5	STMM for luma from X=4 to 7
6	Luma Denoise History for 4x4 at 0,4
7	Not Used
8-15	Repeat for 4x4s at 0,8 and 0,12
16	STMM for 2 luma values at luma Y=1,X=0 to 1
17	STMM for 2 luma values at luma Y=1, X=2 to 3
18	U Chroma Denoise History
19	Not Used
20-31	Repeat for 3 4x4s at 1,4, 1,8 and 1,12
32	STMM for 2 luma values at luma Y=2,X=0 to 1
33	STMM for 2 luma values at luma Y=2, X=2 to 3
34	V Chroma Denoise History
35	Not Used

Byte	Data
36-47	Repeat for 3 4x4s at 2,4, 2,8 and 2,12
48	STMM for 2 luma values at luma Y=3,X=0 to 1
49	STMM for 2 luma values at luma Y=3, X=2 to 3
50-51	Not Used
36-47	Repeat for 3 4x4s at 3,4, 3,8 and 3,12

VEBOX State and Primitive Commands

Every engine can have internal state that can be common and reused across the data entities it processes instead of reloading for every data entity.

There are two kinds of state information:

1. Surface state or state of the input and output data containers.
2. Engine state or the architectural state of the processing unit.

For example in the case of DN/DI, architectural state information such as denoise filter strength can be the same across frames. This section gives the details of both the surface state and engine state.

Each frame should have these commands, in this order:

1. **VEBOX_STATE**
2. **VEBOX_SURFACE_STATE** for input & output
3. **VEB_DI_IECP**

VEBOX State

This chapter discusses various commands that control the internal functions of the VEBOX. The following commands are covered:

- DN/DI State Table Contents
- **VEBOX_IECP_STATE**
- **VEBOX_STATE**
- **VEBOX_Ch_Dir_Filter_Coefficient**

DN-DI State Table Contents

This section contains tables that describe the state commands that are used by the Denoise and Deinterlacer functions.

- **VEBOX_DNDI_STATE**

VEBOX_IECP_STATE

For all piecewise linear functions in the following table, the control points must be monotonically increasing (increasing continuously) from the lowest control point to the highest. Functions which have bias values associated with each control point have the additional restriction that any control points which have the same value must also have the same bias value. The piecewise linear functions include:

- For Skin Tone Detection:
    - Y_point_4 to Y_point_0
    - P3L to P0L
    - P3U to P0U
    - SATP3 to SATP1
    - HUEP3 to HUEP1
    - SATP3_DARK to SATP1_DARK
    - HUEP3_DARK to HUEP1_DARK
- For ACE:
    - Ymax, Y10 to Y1 and Ymin
    - There is no state variable to set the bias for Ymin and Ymax. The biases for these two points are equal to the control point values: B0 = Ymin and B11 = Ymax. That means that if control points adjacent to Ymin and Ymax have the same value as Ymin/Ymax then the biases must also be equal to the Ymin/Ymax control points based on the restriction mentioned above.
    - Forward Gamma correction
    - Gamut Expansion:
        - Gamma Correction
        - Inverse Gamma Correction

- **VEBOX_IECP_STATE**
- **VEBOX_STD_STE_STATE**
- **VEBOX_ACE_LACE_STATE**
- **VEBOX_TCC_STATE**
- **VEBOX_PROCAMP_STATE**
- **VEBOX_CSC_STATE**
- **VEBOX_ALPHA_AOI_STATE**
- **VEBOX_CCM_STATE**
- **Black Level Correction State - DW75..76**

- **VEBOX_FORWARD_GAMMA_CORRECTION_STATE**
- **VEBOX_FRONT_END_CSC_STATE**

For all piecewise linear functions in the following table, the control points must be monotonically increasing (increasing continuously) from the lowest control point to the highest. Any control points which have the same value must also have the same bias value. The piecewise linear functions include:

- PWL_Gamma_Point11 to PWL_Gamma_Point1
- PWL_INV_Gamma_Point11 to PWL_Gamma_Point1
- **VEBOX_GAMUT_STATE**
- **VEBOX_VERTEX_TABLE**
- **VEBOX_CAPTURE_PIPE_STATE**
- **VEBOX_RGB_TO_GAMMA_CORRECTION**

VEBOX Surface State

- **VEBOX_SURFACE_STATE**

Surface Format Restrictions

The surface formats and tiling allowed are restricted, depending on which function is consuming or producing the surface.

Surface Format Restrictions

FourCC Code	Format	DN/DI Input	DN/DI Output	Capture Output	Scalar Input/Output
YUYV	YCRCB_NORMAL (4:2:2)	X	X		
VYUY	YCRCB_SwapUVY (4:2:2)	X	X		
YVYU	YCRCB_SwapUV (4:2:2)	X	X		
UYVY	YCRCB_SwapY (4:2:2)	X	X		
Y8	Y8 Monochrome	X	X		
NV12	NV12 (4:2:0 with interleaved U/V)	X	X		
AYUV	4:4:4 with Alpha (8-bit per channel)				
Y216	4:2:2 packed 16-bit				
Y416	4:4:4 packed 16-bit				
P216	4:2:2 planar 16-bit				
P016	4:2:0 planar 16-bit				
	RGBA 10:10:10:2				
	RGBA 8:8:8:8				
	RGBA 16:16:16:16				
Tiling					
	Tile Y	X	X	X	X
	Tile X	X	X	X	X
	Linear	X	X	X	X

Surface Formats - Feature Notes

Feature
Surfaces are 4 kb aligned, chroma X offset is cache line aligned (16 byte).
If Y8/Y16 is used as the input format, it must also be used for the output format (chroma is not created by VEBOX).
All 16-bit formats are processed at 12-bit internally.

VEB DI IECP Commands

The VEB_DI_IECP command causes the VEBOX to start processing the frames specified by VEB_SURFACE_STATE using the parameters specified by VEB_DI_STATE and VEB_IECP_STATE.

The Surface Control bits for each surface:

- **VEB_DI_IECP**
- **VEB_DI_IECP Command Surface Control Bits**

Command Stream Backend - Video

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this second command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project; the offsets will be maintained.

- VECS_ECOSKPD - VECS ECO Scratch Pad

Video Enhancement Engine Functions

This command streamer supports a completely independent set of registers. Only a subset of the MI Registers is supported for this 2nd command streamer. The effort is to keep the registers at the same offset as the render command streamer registers. The base of the registers for the video decode engine will be defined per project; the offsets will be maintained.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 10: High Efficiency Video Coding (HEVC)

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

High Efficiency Video Coding (HEVC) Introduction 1
Scope 1
Summary of Features 1
HCP Hardware Pipeline Features 2
HEVC Decoder Features 3
HCP Command Summary 3
Workload Command Model 4
HCP Command Sequence Examples 5
HCP Decoder Command Sequence 5
Memory Address Attributes 6
HCP Pipe Common Commands 6
Buffer Size Requirements 7
HCP Common Commands 8
HCP Commands 8
HEVC Error Concealment 9
HEVC Register Definitions 11
Register Attributes Description 11
HCP Decoder Register Map 11
HCP Decoder Register Descriptions 12
Acronyms and Abbreviations 13

High Efficiency Video Coding (HEVC) Introduction

The HEVC Codec Pipeline (HCP) is a fixed function hardware video codec responsible for decoding HEVC (High Efficiency Video Coding) video streams.

Scope

The primary scope of the HCP BSpec document is to provide a description of the HCP commands processed by the Video Command Streamer (VCS). The secondary scope is to provide a description of the status registers on the Message Channel Interface to support decoding of the HEVC video format.

The BSpec sections include:

- Summary of Features
- Architecture Overview
- Commands
- Register Definitions
- Acronyms and Applicable Standards

Summary of Features

The following sections define the HEVC Decoder general features, and the features specific to HEVC decoding.

HCP Hardware Pipeline Features

- Supports decoder functions, setup on a per picture basis:
  - Hardware acceleration provides Ctb/CU level decode.
  - No context switch is supported within a frame process.
- Supports Video Command Streamer (VCS):
  - Shared with MFX HW pipeline, and at any one time, only one pipeline (MFX or HCP) and one operation (decoding or encoding) can be active.
- Supports Message Channel Interface:

Feature
Supports Tile-Y Legacy.

- Supports NV12 video buffer plane:
  - Supports 4:2:0, 8-bit per pixel component (Y, Cb and Cr) video.
- Supports 8Kx8K frame size.

HEVC Decoder Features

- Supports full-featured HEVC Main Profile standard, up to Level 6.2.
- Supports the long format HW decoding interface:
  - All headers (SPS, PPS, Slice Header) are parsed and decoded outside the HCP HW pipeline. They are then fed to the HW through a set of HCP state commands.
- Error detection/resiliency down to the Ctb/CU level.

HCP Command Summary

The HCP is configured for decoding through a set of batch commands defined in the following sections. The software driver builds a frame level workload using these commands and stores these workloads in graphics memory where they are fetched by the Video Command Streamer (VCS) and presented to the HCP for processing. The commands are processed by the Workload Parser within the HCP and the hardware is configured by the Workload Parser prior to each frame level decode. A workload is defined as a set of commands necessary to decode one frame.

The software driver is required to read the HCP disable fuse to determine if the HCP is enabled. If it is disabled, then the software driver must not enable HCP batch commands to be sent to the HCP or a hang event may occur. Only when the HCP is enabled through the fuse, should the batch commands be sent to the HCP.

Workload Command Model

DWord0 of each command is defined in HCP DWord0 Command Definition. The HCP is selected with the **Media Instruction Opcode "7h**" for all HCP Commands.

HCP DWord0 Command Definition

DWord	Bits	Description
0	31:29	**Command Type** = PARALLEL_VIDEO_PIPE = 3h
	28:27	**Pipeline Type** = 2h
	26:23	**Media Instruction Opcode = Codec/Engine Name** = HCP = 7h
	22:16	**Media Instruction Command** = *<see Workload Command Model>*
	15:12	**Reserved: MBZ**
	11:0	**Dword Length** (Excludes Dwords 0, 1) = *<command length>*

Each HCP command has assigned a media instruction command as defined in HCP Media Instruction Commands (Opcode=7h).

HCP Media Instruction Commands (Opcode=7h)

Media Instruction Command	Command DWord0 [22:16]	Mode	Scope
HCP_PIPE_MODE_SELECT	0h	Dec	Picture
HCP_SURFACE_STATE	1h	Dec	Picture
HCP_PIPE_BUF_ADDR_STATE	2h	Dec	Picture
HCP_IND_OBJ_BASE_ADDR_STATE	3h	Dec	Picture
HCP_QM_STATE	4h	Dec	Picture
Reserved	5h-Fh		
HCP_PIC_STATE	10h	Dec	Picture
HCP_TILE_STATE	11h	Dec	Picture
HCP_REF_IDX_STATE	12h	Dec	Slice
HCP_WEIGHTOFFSET	13h	Dec	Slice
HCP_SLICE_STATE	14h	Dec	Slice
Reserved	15h-1Fh		
Reserved	20h	Dec	Slice
Reserved	21h		
Reserved	22h		
Reserved	23h-7Fh		

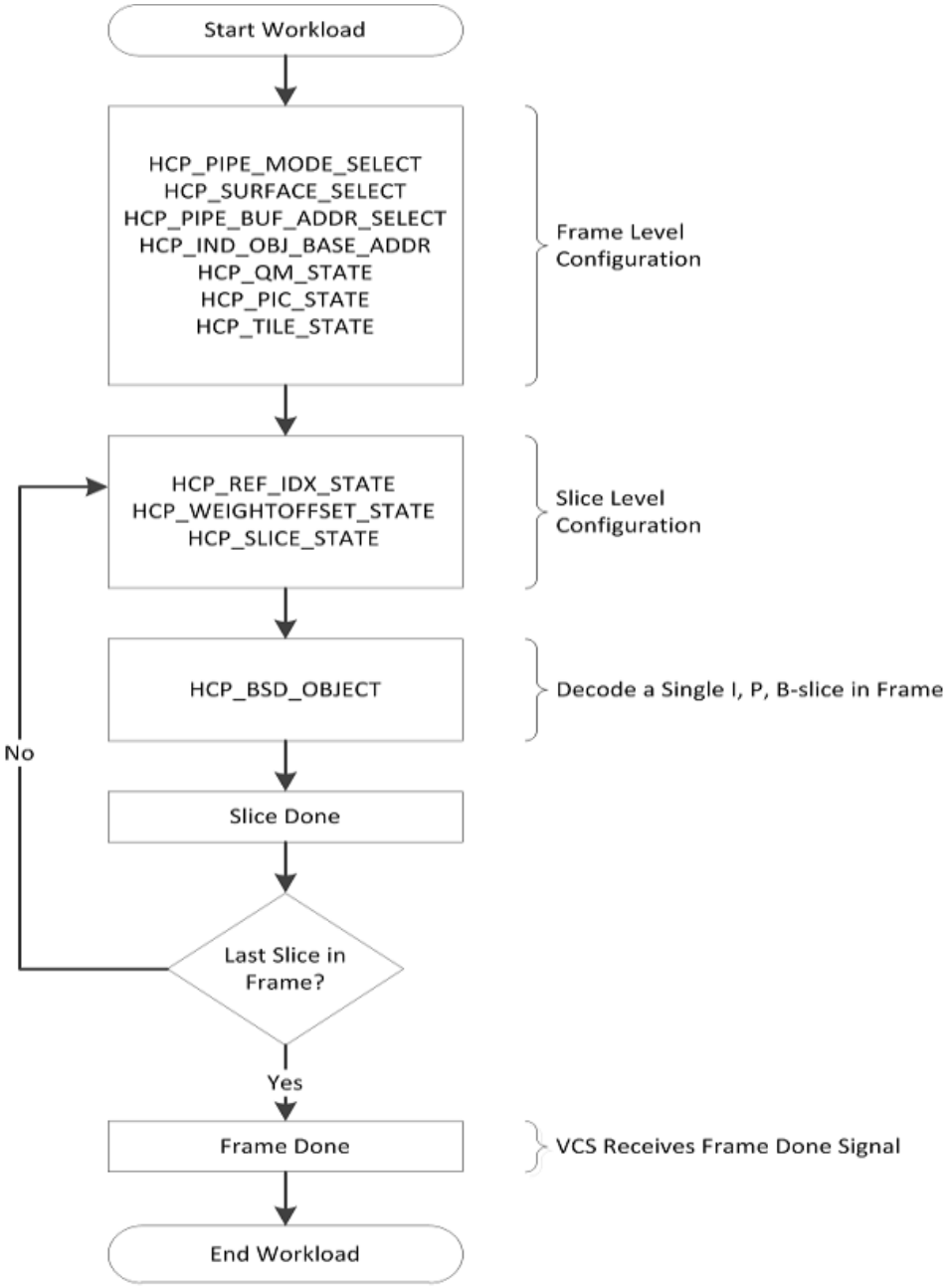
HCP Command Sequence Examples

This section is currently under development.

HCP Decoder Command Sequence

The long format workload for the HCP is based upon a single frame decode. There are no states saved between frame decodes in the HCP. Once the bit stream DMA is configured with the HCP_BSD_OBJECT command, and the bit stream is presented to the HCP, the frame decode will begin.

HCP Long Format Decode Workload Flowchart



Memory Address Attributes

This section defines the memory address attributes for the third DWord of the HCP command buffer address.

NOTE: The first DWord defines the lower address range and the second Dword defines the upper address range in the HCP command buffer address.

MemoryAddressAttributes

HCP Pipe Common Commands

The HCP Pipe Common Commands specify the HEVC Decoder pipeline level configuration.

HCP_PIPE_MODE_SELECT (VideoCS)

HCP_SURFACE_STATE (VideoCS)

HCP_PIPE_BUF_ADDR_STATE (VideoCS)

HCP_IND_OBJ_BASE_ADDR_STATE (VideoCS)

HCP_QM_STATE (VideoCS)

Buffer Size Requirements

HEVC Buffer Size Requirements

Buffer Name	Minimum Size in CLs	Notes
Deblocking Filter Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Line Buffer	((picture_width_in_pixels + 31) & (-32)) » 3	Eq. ensures multiple of 4
Deblocking Filter Tile Column Buffer	((picture_height_in_pixels + 6*pic_height_in_ctb + 31) & (-32)) » 3	Eq. ensures multiple of 4
Metadata Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu*8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (all intra slices)	(picture_width_in_pixels + pic_width_in_lcu *8 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (all intra slices)	(picture_height_in_pixels + picture_height_in_lcu * 16 + 1023) » 9	Eq. ensures multiple of 2
Metadata Line Buffer (some inter slices)	(((picture_width_in_pixels+15)»4)*188 + pic_width_in_lcu * 9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Line Buffer (some inter slices)	(((picture_width_in_pixels +15)»4)*172 + pic_width_in_lcu * 9 + 1023) » 9	Eq. ensures multiple of 2
Metadata Tile Column Buffer (some inter slices)	(((picture_height_in_pixels +15)»4)*256 + picture_height_in_lcu * 9 + 1023) » 9	Eq. ensures multiple of 2
SAO Line Buffer	(((picture_width_in_pixels » 1) + pic_width_in_ctb * 3)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Line Buffer	(((picture_width_in_pixels » 1) + picture_width_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2
SAO Tile Column Buffer	(((picture_height_in_pixels » 1) + pic_height_in_ctb * 6)+15) & (-16)) » 3	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu=16x16)	(((picture_width_in_pixels+63)»6)*((picture_height_in_pixels+15)»4)	Eq. ensures multiple of 2
Current and Collocated Motion Vector Temporal Buffer (lcu>16x16)	(((picture_width_in_pixels+31)»5)*((picture_height_in_pixels+31)»5)	Eq. ensures multiple of 2
SSE Line Buffer	(Picture_width_in_lcu + 2) « 4	

HCP Common Commands

HCP_PIC_STATE (VideoCS)

HCP_TILE_STATE (VideoCS)

HCP_REF_IDX_STATE (VideoCS)

HCP_WEIGHTOFFSET_STATE (VideoCS)

HCP_SLICE_STATE (VideoCS)

HCP Commands

The HCP Commands specify the HEVC BSD object.

HCP_BSD_OBJECT (VideoCS)

HEVC Error Concealment

The HCP implements an error concealment policy, which is always enabled and cannot be disabled. The objective is that the HCP will always complete a frame/field workload by either decoding the bit stream normally until it finishes the workload or by concealing blocks until the slice or workload is completed. It should never be allowed to hang.

Error concealment, implemented by the HCP hardware, is configured for each slice in the HCP_BSD_OBJECT command. The following information in the HCP_BSD_OBJECT command is utilized for error concealment.

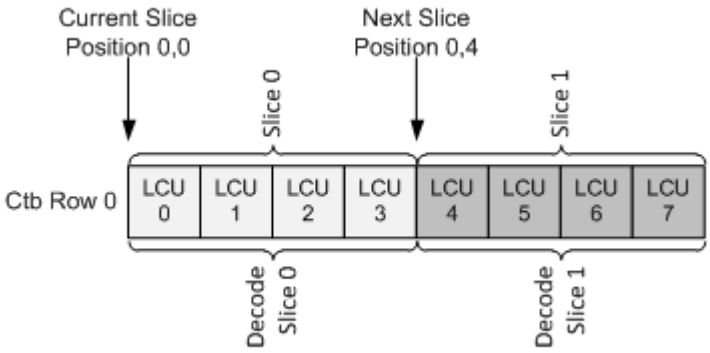
- **SliceStartCtbY, SliceStartCtbX:** The current slice position specified in Ctb coordinates.
- **NextSliceStartCtbY, NextSliceStartCtbX:** The next slice position specified in Ctb coordinates. If the current slice is the last slice in the picture, the next slice values are set to (0,0).
- **LastSliceofPic:** Indicates that the current slice is the last slice in the picture.
- **slice_type:** Indicates the picture type: I, P or B.

The host software will remove all extra slices in the picture. The HCP will not be given a workload that includes extra slices beyond the picture. The last slice in the picture will always be marked by the host software.

The host software will remove any overlapping slices in the picture. The HCP will not be given a workload that includes overlapping slices in the picture.

A HCP_BSD_OBJECT command will include the current slice position and the next slice position. For non-errored streams, it is guaranteed that the slice bit stream will be decoded by the HCP starting from the current slice position through to the Ctb (inclusive) adjacent to the Ctb indicated by the next slice position. *HEVC Error Concealment* illustrates the example of a non-errored stream decode starting with XXX.

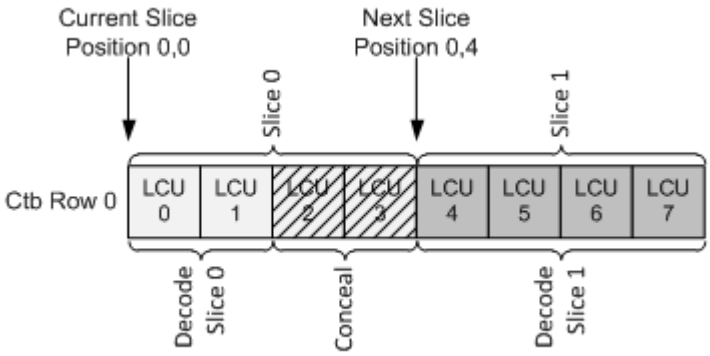
HEVC Slice Decode for Non-errored Stream Cases



For error stream cases where the next slice position does not align itself with the last successfully decoded Ctb in the current slice, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb prior to the Ctb indicated by the next slice position. If the error occurs such that the current decoded Ctb cannot be decoded, the HCP will ensure that the current Ctb is written out

by any means before writing out concealed Ctbs for the remaining Ctbs in the current slice. In the case of the last slice in a picture, the HCP will conceal Ctbs from the last decoded Ctb in the current slice through to the last Ctb position in the picture indicated by the resolution of the picture in the HCP_PICT_STATE command. *HEVC Error Concealment* illustrates the case described.

HEVC Slice Decode for Missing Blocks in a Slice



Since the host software removes overlapping slices, the next slice position will never be equal to or less than the current slice position.

A concealed Ctb for an I-slice is constructed by the HCP specifying the Intra_Planar prediction mode for the Ctb.

A concealed Ctb for a P-slice is constructed by the HCP specifying the skip_flag.

A concealed Ctb for a B-slice is constructed by the HCP specifying the skip_flag.

HEVC Register Definitions

The Message Channel Interface is a read-only bus used to access the HCP status register. All registers are 32 bits, where reserved bits return a value of zero and subtractive-decode is used to return 0x0000 for all register holes. The Unit ID is 28h. For HCP, the address range is 0x0001E900h to 0001E9FFh.

Register Attributes Description

Host Register Attributes gives the defined register tags and their description.

Host Register Attributes

Tag	Name	Description
R/W	Read/Write	Bit is read and writeable.
R/SW	Read/Special Write	Bit is readable. Write is only allowed once after a reset.
RO	Read Only	Bit is only readable, but writes have no effects.
WO	Write Only	Bit is only writeable, reads return zeros.
RV	Reserved	Bit is reserved and not visible. Reads will return 0, and writes have no effect.
NA	Not Accessible	This bit is not accessible.

HCP Decoder Register Map

Register	Address	Default	Description
HCP_DEC_STATUS	1E900h	00000000H	HCP decode status
HCP_CABAC_STATUS	1E904h	00000000H	HCP CABAC status
HCP_LAST_POSITION	1E908h	00000000H	Last row & column position of the decoder
HCP_PMU_STATUS	1E90Ch	00000000H	PMU counter overflow status
HCP_PMU_LUMA_CNTR	1E910h	00000000H	Luma cache miss event counter
HCP_PMU_CHROMA_CNTR	1E914h	00000000H	Chroma cache miss event counter
HCP_PMU_FRAME_DEC_ACT_CNTR	1E918h	00000000H	Frame decode active event counter
HCP_PICTURE_CHECKSUM_CIDX0	1E91Ch	00000000h	Picture checksum cIdx0
HCP_PICTURE_CHECKSUM_CIDX1	1E920h	00000000h	Picture checksum cIdx1
HCP_PICTURE_CHECKSUM_CIDX2	1E924h	00000000h	Picture checksum cIdx2

HCP Decoder Register Descriptions

The HCP implements the following MMIO registers. A description of the register including its address and DWord descriptions are provided.

HCPDecodeStatus (VideoCS)

HCPCABACStatus (VideoCS)

HCPLastPosition (VideoCS)

HCPPMUStatus (VideoCS)

HCPPMULumaCacheMissCounter (VideoCS)

HCPPMUChromaCacheMissCounter (VideoCS)

HCPPMUFrameDecodeActiveCounter (VideoCS)

HCPPictureChecksumcIdx0 (VideoCS)

HCPPictureChecksumcIdx1 (VideoCS)

HCPPictureChecksumcIdx2 (VideoCS)

Acronyms and Abbreviations

The table below defines acronyms and abbreviations used in this document.

Acronyms

Acronym	Meaning
AAC	Advanced Audio Coding — part of the MPEG specification, AAC is the latest development in audio compression. It provides higher-quality audio reproduction than MPEG-1 Layer 3 (MP3), while requiring nearly 50% less data. It is defined in ISO/IEC 13818-7.
ADSL	Asymmetrical Digital Subscriber Line — an asymmetrical DSL technology that takes advantage of the one-way nature of most multimedia communication, and provides much faster data rates for downstream (to the subscriber) then the upstream.
API	Application Programming Interface — a set of routines used by an application program to request and carry out low-level services performed by the operating system.
ARGB	Alpha Red Green Blue — color channel components.
ARIB	Association of Radio Industries and Business — designated by the Ministry of Public Management, Home Affairs, Posts and Telecommunications (MPHPT) in Japan. ARIB members include broadcasters, radio equipment manufacturers, telecommunication operators, and related organizations.
ASP	Advanced Simple Profile – MPEG4-2
ATSC	ATSC Advanced Television Systems Committee – an organization in US that establishes and promotes technical standards for advanced television systems, such as digital television (DTV).
BDU	Bit-stream Data Unit
BIST	Built In Self Test
BPP	Bits Per Pixel
BSD	Byte Stream Decoder
CA, CAM	Conditional Access, Conditional Access Module – the removable descrambling module implemented in digital cable or satellite television system. The data flows through the module, which can have any proprietary scrambling algorithm implemented, yet maintaining system interface compatibility. The CAMs are usually provided by the operators in the TV network.
CPU	Central Processing Unit
DAA	Direct Access Arrangement
DAC	Digital-to-Analog Converter
DDA	Digital Difference Analyzer
DDS	Direct Digital Synthesizer
DPB	Decoded Picture Buffer. This buffer holds the decoded pictures for reference and for output along with the currently decoding picture. This differs from the DPB in the standard, which only holds the decoded pictures for reference.
DVB	Digital Video Broadcasting — a set of open worldwide standards that define digital broadcasting using existing satellite, cable, and terrestrial infrastructures. It uses MPEG-2 specification as a universal foundation and expands it with DVB data structures and processes DVB-compliant digital broadcasting and equipment is widely available to consumers and is indicated with the DVB

Acronym	Meaning
	logo.
DVB-S	Satellite television DVB standards, based on QPSK and 8-DPSK modulation.
DVB-T	Terrestrial television DVB standards, based on 2k and 8k OFDM modulation.
DVD	Digital Versatile Disc
DVD-R	Recordable DVD. Since different disk formats are currently in use, including DVD-R,DVD+R, they are collectively mentioned as DVD-R in this document
DVI	Digital Visual Interface standard (EIA/CEA-861A). The standard defines a method for sending digital video signals over DVI and OpenLDI interface specifications. The standard is fully backward compatible with earlier DVI standards. New features include carrying auxiliary video information, such as aspect ratio and native video format information.
DSL xDSL	Digital Subscriber Line – transmission of data over copper telephone lines capable of bringing high-bandwidth to subscribers. Many flavors of DSL are currently in use, which are collectively called xDSL throughout the document.
DSP	Digital Signal Processor
DST	Destination
DWord	A 32-bit word
FIFO	First in First Out
FIR	Finite Impulse Response
FPU	Floating Point Unit
FW	Firmware running on the decoder controller, as used in Volume 4 of the *Olo River Plus Silicon EAS*
IDR	Instantaneous Decoding Refresh
IEEE 1394 1394	IEEE 1394 or iLink* or FireWire* An IEEE electronics industry standard for connecting multimedia and computing Up to 63 devices can be attached to your PC via a single plug-and-socket connection.
IEEE 802.11 802.11	The Institute for Electronics and Electrical Engineers (IEEE) wireless network specification. 802.11g and 802.11a networks can transmit payload at the rates in excess 34Mbits/s and allow for the wireless transmission at distances from several dozen to several hundred feet indoors.
IF	Intermediate Frequency — the fixed, relatively low-frequency carrier to which current programs are ported by the tuner.
GMCH	Graphics and Memory Control Hub — a chip that connects the IA processor to memory and other components in PC.
HDD	Hard Disk Drive — magnetic mass storage device used in media centers for audiovisual program recording.
HDMI	High Definition Multimedia Interface (HDMI). This interface is used between any audio/video source, such as a set-top box, DVD player, or A/V receiver, and an audio or video monitor, such as a DTV. HDMI supports standard, enhanced or high-definition video, plus multi-channel digital audio on a single cable. The format transmits all ATSC HDTV standards and supports eight-channel digital audio (at up to a 192kHz sampling rate), with bandwidth to spare for future enhancements.

Acronym	Meaning
HDTV	High-Definition Television — HDTV specifically refers to the highest-resolution formats of the 18 total DTV formats, true HDTV is generally considered to be 1,080-line interlaced (1080i) or 720-line progressive (720p).
HSR	Hidden Surface Removal
HW	Hardware
I/F	Interface
IEEE	IEEE 32-bit Floating Point number format representation
ISP	Image Synthesis Processor — A collective term to describe all components of the hidden surface removal operation within the PowerVR architecture.
LOD	Level Of Detail — used in texturing calculations.
LSB	Least Significant Bit
LUT	Look-up table
MBAFF	Block Adaptive Field Frame mode
MFD	Multi-Format Decoder
MMU	Memory Management Unit
MMMC	Multi-port, Multi-channel Memory Controller
MSA	Intel Micro Signal Architecture — microprocessor architecture combining the features of microcontroller and digital signal processor. MSA is used here as a synonym of the processor core used in Olo River Plus
MSB	Most Significant Bit
MPEG	Motion Picture Experts Group – Organization that develops standards for digital video and digital audio compression.
MPR	Inter Prediction Module
NAL	Network Abstraction Layer
NAL unit	Syntax structure in a H.264 stream
NTSC	National Television System Committee, North American 525-line analog broadcast TV standard.
NIM	Network Interface Module – the integrated tuner and digital demodulator in the (satellite) TV systems. The DVB NIMs output MPEG transport stream.
NOP	No operation
OEM	Original Equipment Manufacturer
OGL/OpenGL	Open GL application programming interface
PAL	Phase Alternation Line - TV standard used in Europe. PAL uses 625 lines per frame, a 25 frames per second update rate and YUV color encoding. The number of visible pixels for PAL video is 768 x 576.
PCI	Peripheral Component Interconnect bus, a bi-directional bus defined in PCI 2.x specification
PES	Packetized Elementary Streams — packetized streams are the ES streams arranged in data packets with PES header starting every packet. The syntax of the ES and PES is defined in MPEG. See definition for ES.
PIP	Picture In Picture display mode

Acronym	Meaning
POD	Point of Deployment conditional access module — the removable conditional access module defined in the OpenCable* specification in US.
PPS	Picture Parameter set
PTS	Presentation time stamp
PVR	Personal Video Recorder, also PDR or personal digital recorder — an interactive TV-recording device that records programs in digital format and allows users to search for/record shows based on type (for instance all basketball games or all episodes of a particular program). Users can also pause, rewind, stop, or fast-forward live programs with only a small time lag.
PWL	Piece-wise Linear
PXD	Pixel Decoder Module
RF	Radio Frequency – usually, modulated carriers which can be directly received by the tuners of TVs or radio receivers
RISC	Reduced Instruction Set Computer
RHW	Reciprocal Homogenous W — W is a 3-D coordinate representation like X Y Z
RSB	Row Store Buffer
RTL	Register Transfer Language/Level
SEI	Supplementary Enhancement Information
SIF	Semaphore Interface Module
SIMD	Single Instruction Multiple Data
SMPTE	Society of Motion Picture and Television Engineers
SOC	System on chip
SP	Simple Profile – MPEG4-2
SPS	Sequence Parameter set
SRC	Source
SDTV	Standard-Definition Television — a digital television system that is similar to current analog TV standards in picture resolution and aspect ratio. Typical SDTV resolution is 480i or 480p.
STB	Set Top Box — a device that effectively turns a television set into an interactive Internet device and/or allows the television to receive and decode digital television (DTV) broadcasts.
TA	Tile Accelerator
TS	MPEG-2 Transport Stream — a sequence of 188-byte packets carrying the multi-program audiovisual data
TSP	Texture Shading Processor — a collective term to describe all components of the texture, shading and pixel blending operations within the PowerVR architecture.
VCL	Video Coded Layer
VCXO	Voltage Controlled Crystal Oscillator
VGP/ VGP Lite	Vertex Geometry Processor
VLC	Variable length coded. This refers to the collection of coding techniques that are used in VC1, and include CABAC, CAVLC and Exp-Golomb.
VOL	Video Object Layer

Acronym	Meaning
VOP	Video Object Plane
WAN	Wide Area Network
WSS	Wide Screen Signaling
XDS	Extended Data Services — data services sending data in line 21/283 of the analog NTSC TV signal
XSI	Intel® XScale® System Interconnect
X, Y, Z, W	3-D coordinate representations
YUV	YUV texture format, primarily for video formats

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 11: Blitter

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Blitter (BLT) Engine Command Streamer 1
Software Control Bit Definitions 1
Registers for Blitter Engine Command Streamer 1
BLT Engine 2
Introduction 2
Classical BLT Engine Functional Description 3
Basic BLT Functional Considerations 5
Color Depth Configuration and Color Expansion 5
Graphics Data Size Limitations 5
Bit-Wise Operations 6
Per-Pixel Write-Masking Operations 10
When the Source and Destination Locations Overlap 12
Basic Graphics Data Considerations 16
Contiguous vs. Discontinuous Graphics Data 16
Source Data 18
Monochrome Source Data 18
Pattern Data 19
Destination Data 23
BLT Programming Examples 24
Pattern Fill — A Very Simple BLT 24
Drawing Characters Using a Font Stored in System Memory 27
BLT Instruction Overview 31
BLT Engine State 31
Cacheable Memory Support 32
Device Cache Coherency: Render & Texture Caches 32
BLT Engine Instructions 32
BLT Programming Restrictions 33
2D (XY) BLT Instructions 34
BLT Engine Instruction Field Definitions 37

Blitter (BLT) Engine Command Streamer

This pipeline has its own command streamer and operates completely independently of the other command streamers. This command streamer supports a separate set of registers starting at offset 20000h.

Software Control Bit Definitions

Registers in the range 22XX are not protected from the load register immediate instruction if the command is executed in the non-secure batch buffer.

BCS_SWCTRL - BCS SW Control

BCS_TIMESTAMP - BCS Reported Timestamp Count

BCS_CTX_TIMESTAMP - BCS Context Timestamp Count

Registers for Blitter Engine Command Streamer

These are the Registers for the Blitter Engine Command Streamer.

GAB_MODE - Mode Register for GAB

BLT Engine

Introduction

2D Rendering can be divided into 2 categories: classical BLTs, described here, and 3D BLTs. 3D BLTs are operations which can take advantage of the 3D drawing engine's functionality and access patterns.

Functions such as Alpha BLTs, arithmetic (bilinear) stretch BLTs, rotations, transposing pixel maps, color space conversion, and DIBs are all considered 3D BLTs and are covered in the 3D rendering section. DIBs can be thought of as an indexed texture which uses the texture palette for performing the data translation. All drawing engines have swappable context. The same hardware can be used by multiple driver threads where the current state of the hardware is saved to memory and the appropriate state is loaded from memory on thread switches.

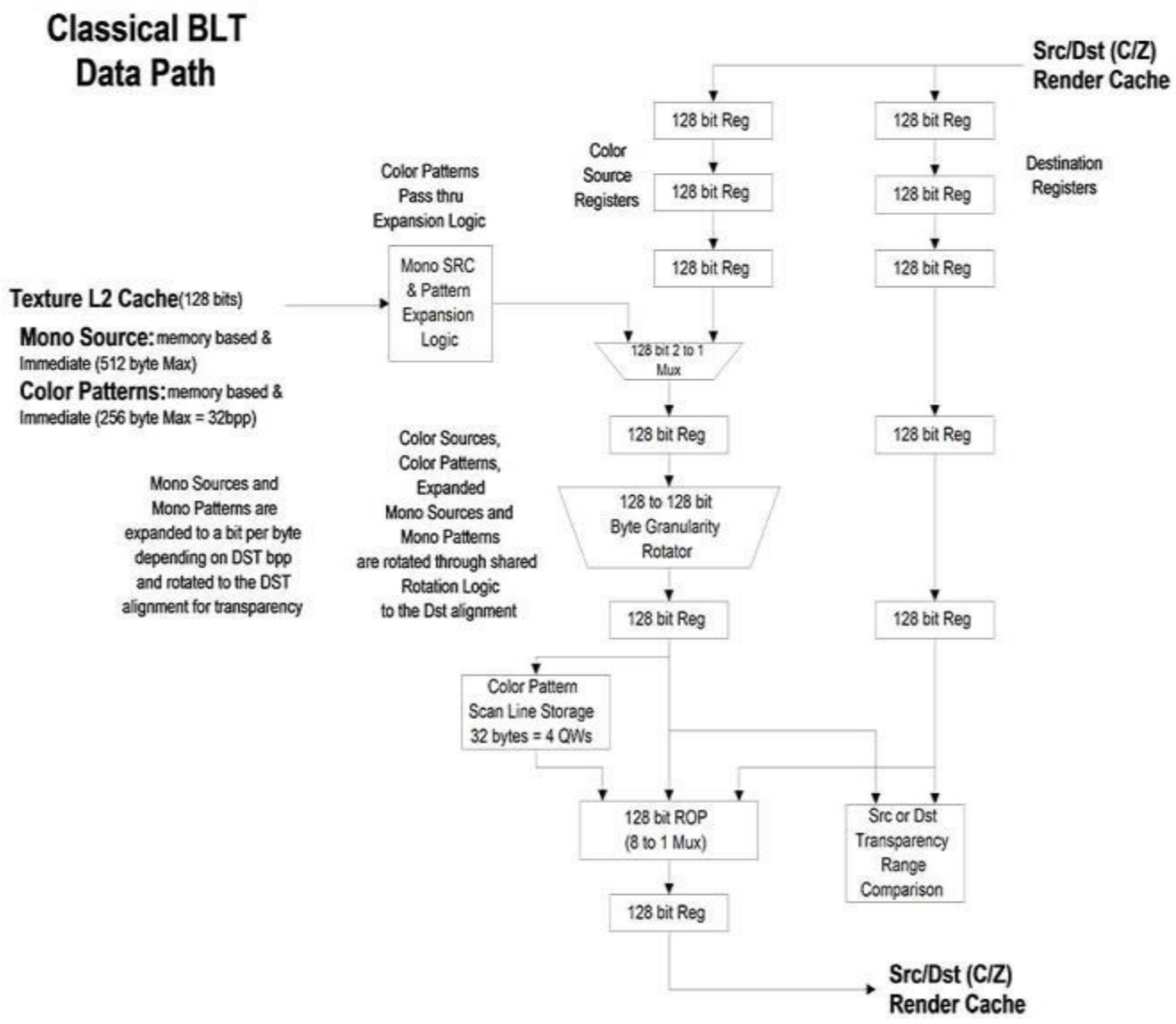
All operands for both 3D and classical BLTs can be in graphics aperture or cacheable system memory. Some operands can be immediates which are sent through the command stream. Immediate operands are: patterns, monochrome sources, DIB palettes, and DIB source operands. All non-monochrome operands which are not tiled have a stride granularity of a double-word (4 bytes).

The classical BLT commands support both linear addressing and X, Y coordinates with and without clipping. All X1 and Y1 destination and clipping coordinates are inclusive, while X2 and Y2 are exclusive. Currently, only destination coordinates can be negative. The source and clipping coordinates must be positive. If clipping is disabled, but a negative destination coordinate is specified, the negative coordinate is clipped to 0. Linear address BLT commands must supply a non-zero height and width. If either height or width = 0, then no accesses occur.

Linear address (or non-XY) BLT commands are not supported.

Classical BLT Engine Functional Description

The graphics controller provides a hardware-based BLT engine to off load the work of moving blocks of graphics data from the host CPU. Although the BLT engine is often used simply to copy a block of graphics data from the source to the destination, it also has the ability to perform more complex functions. The BLT engine is capable of receiving three different blocks of graphics data as input as shown in the figure below. The source data may exist in the frame buffer or the Graphics aperture. The pattern data always represents an 8x8 block of pixels that can be located in the frame buffer, Graphics aperture, or passed through a command packet. The pattern data must be located in linear memory. The data already residing at the destination may also be used as an input. The destination data can also be located in the frame buffer or graphics aperture.



Block Diagram and Data Paths of the BLT Engine

The BLT engine may use any combination of these three different blocks of graphics data as operands, in both bit-wise logical operations to generate the actual data to be written to the destination, and in per-pixel write-masking to control the writing of data to the destination. It is intended that the BLT engine will perform these bit-wise and per-pixel operations on color graphics data that is at the same color depth that the rest of the graphics system has been set. However, if either the source or pattern data is monochrome, the BLT engine has the ability to put either block of graphics data through a process called “color expansion” that converts monochrome graphics data to color. Since the destination is often a location in the on-screen portion of the frame buffer, it is assumed that any data already at the destination will be of the appropriate color depth.

Basic BLT Functional Considerations

This topic is currently under development.

Color Depth Configuration and Color Expansion

The graphics system and BLT engine can be configured for color depths of 8, 16, and 32 bits per pixel.

The configuration of the BLT engine for a given color depth dictates the number of bytes of graphics data that the BLT engine will read and write for each pixel while performing a BLT operation. It is assumed that any graphics data already residing at the destination which is used as an input is already at the color depth to which the BLT engine is configured. Similarly, it is assumed that any source or pattern data used as an input has this same color depth, unless one or both is monochrome. If either the source or pattern data is monochrome, the BLT engine performs a process called "color expansion" to convert such monochrome data to color at the color depth to which the BLT engine has been set.

During "color expansion" the individual bits of monochrome source or pattern data that correspond to individual pixels are converted into 1, 2, or 4 bytes (which ever is appropriate for the color depth to which the BLT engine has been set). If a given bit of monochrome source or pattern data carries a value of 1, then the byte(s) of color data resulting from the conversion process are set to carry the value of a specified foreground color. If a given bit of monochrome source or pattern data carries a value of 0, the resulting byte(s) are set to the value of a specified background color or not written if transparency is selected.

The BLT engine is set to a default configuration color depth of 8, 16, or 32 bits per pixel through BLT command packets. Whether the source and pattern data are color or monochrome must be specified using command packets. Foreground and background colors for the color expansion of both monochrome source and pattern data are also specified through the command packets. The source foreground and background colors used in the color expansion of monochrome source data are specified independently of those used for the color expansion of monochrome pattern data.

Graphics Data Size Limitations

The BLT engine is capable of transferring very large quantities of graphics data. Any graphics data read from and written to the destination is permitted to represent a number of pixels that occupies up to 65,536 scan lines and up to 32,768 bytes per scan line at the destination. The maximum number of pixels that may be represented per scan line's worth of graphics data depends on the color depth.

Any source data used as an input must represent the same number of pixels as is represented by any data read from or written to the destination, and it must be organized so as to occupy the same number of scan lines and pixels per scan line.

The actual number of scan lines and bytes per scan line required to accommodate data read from or written to the destination are set in the destination width & height registers or using X and Y coordinates within the command packets. These two values are essential in the programming of the BLT engine, because the engine uses these two values to determine when a given BLT operation has been completed.

Bit-Wise Operations

The BLT engine can perform any one of 256 possible bit-wise operations using various combinations of the three previously described blocks of graphics data that the BLT engine can receive as input.

The choice of bit-wise operation selects which of the three inputs will be used, as well as the particular logical operation to be performed on corresponding bits from each of the selected inputs. The BLT engine automatically foregoes reading any form of graphics data that has not been specified as an input by the choice of bit-wise operation. An 8-bit code written to the raster operation field of the command packets chooses the bit-wise operation. The following table lists the available bit-wise operations and their corresponding 8-bit codes.

Bit-Wise Operations and 8-Bit Codes (00-3F)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
00	writes all 0's	20	D and (P and (notS))
01	not(D or (P or S)))	21	not(S or(D xor P))
02	D and (not(P or S))	22	D and (notS)
03	not(P or S)	23	not(S or (P and (notD)))
04	S and (not(D or P))	24	(S xor P) and (D xor S)
05	not(D or P)	25	not(P xor (D and (not(S and P))))
06	not(P or (not(D xor S)))	26	S xor (D or (P and S))
07	not(P or (D and S))	27	S xor (D or (not(P xor S)))
08	S and (D and (notP))	28	D and (P xor S)
09	not(P or (D xor S))	29	not(P xor (S xor (D or (P and S))))
0A	D and (notP)	2A	D and (not(P and S))
0B	not(P or (S and (notD)))	2B	not(S xor ((S xor P) and (P xor D)))
0C	S and (notP)	2C	S xor (P and (D or S))
0D	not(P or (D and (notS)))	2D	P xor (S or (notD))
0E	not(P or (not(D or S)))	2E	P xor (S or (D xor P))
0F	notP	2F	not(P and (S or (notD)))
10	P and (not(D or S))	30	P and (notS)
11	not(D or S)	31	not(S or (D and (notP)))
12	not(S or (not(D xor P)))	32	S xor (D or (P or S))
13	not(S or (D and P))	33	notS
14	not(D or (not(P xor S)))	34	S xor (P or (D and S))
15	not(D or (P and S))	35	S xor (P or (not(D xor S)))
16	P xor (S xor (D and (not(P and S))))	36	S xor (D or P)
17	not(S xor ((S xor P) and (D xor S)))	37	not(S and (D or P))
18	(S xor P) and (P xor D)	38	P xor (S and (D or P))
19	not(S xor (D and (not(P and S))))	39	S xor (P or (notD))

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
1A	P xor (D or (S and P))	3A	S xor (P or (D xor S))
1B	not(S xor (D and (P xor S)))	3B	not(S and (P or (notD)))
1C	P xor (S or (D and P))	3C	P xor S
1D	not(D xor (S and (P xor D)))	3D	S xor (P or (not(D or S)))
1E	P xor (D or S)	3E	S xor (P or (D and (notS)))
1F	not(P and (D or S))	3F	not(P and S)

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (40 - 7F)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
40	P and (S and (notD))	60	P and (D xor S)
41	not(D or (P xor S))	61	not(D xor (S xor (P or (D and S))))
42	(S xor D) and (P xor D)	62	D xor (S and (P or D))
43	not(S xor (P and (not(D and S))))	63	S xor (D or (notP))
44	S and (notD)	64	S xor (D and (P or S))
45	not(D or (P and (notS)))	65	D xor (S or (notP))
46	D xor (S or (P and D))	66	D xor S
47	not(P xor (S and (D xor P)))	67	S xor (D or (not(P or S)))
48	S and (D xor P)	68	not(D xor (S xor (P or (not(D or S)))))
49	not(P xor (D xor (S or (P and D))))	69	not(P xor (D xor S))
4A	D xor (P and (S or D))	6A	D xor (P and S)
4B	P xor (D or (notS))	6B	not(P xor (S xor (D and (P or S))))
4C	S and (not(D and P))	6C	S xor (D and P)
4D	not(S xor ((S xor P) or (D xor S)))	6D	not(P xor (D xor (S and (P or D))))
4E	P xor (D or (S xor P))	6E	S xor (D and (P or (notS)))
4F	not(P and (D or (notS)))	6F	not(P and (not(D xor S)))
50	P and (notD)	70	P and (not(D and S))
51	not(D or (S and (notP)))	71	not(S xor ((S xor D) and (P xor D)))
52	D xor (P or (S and D))	72	S xor (D or (P xor S))
53	not(S xor (P and (D xor S)))	73	not(S and (D or (notP)))
54	not(D or (not(P or S)))	74	D xor (S or (P xor D))
55	notD	75	not(D and (S or (notP)))
56	D xor (P or S)	76	S xor (D or (P and (notS)))

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
57	not(D and (P or S))	77	not(D and S)
58	P xor (D and (S or P))	78	P xor (D and S)
59	D xor (P or (notS))	79	not(D xor (S xor (P and (D or S))))
5A	D xor P	7A	D xor (P and (S or (notD)))
5B	D xor (P or (not(S or D)))	7B	not(S and (not(D xor P)))
5C	D xor (P or (S xor D))	7C	S xor (P and (D or (notS)))
5D	not(D and (P or (notS)))	7D	not(D and (not(P xor S)))
5E	D xor (P or (S and (notD)))	7E	(S xor P) or (D xor S)
5F	not(D and P)	7F	not(D and (P and S))

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (80 - BF)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
80	D and (P and S)	A0	D and P
81	not((S xor P) or (D xor S))	A1	not(P xor (D or (S and (notP))))
82	D and (not(P xor S))	A2	D and (P or (notS))
83	not(S xor (P and (D or (notS))))	A3	not(D xor (P or (S xor D)))
84	S and (not(D xor P))	A4	not(P xor (D or (not(S or P))))
85	not(P xor (D and (S or (notP))))	A5	not(P xor D)
86	D xor (S xor (P and (D or S)))	A6	D xor (S and (notP))
87	not(P xor (D and S))	A7	not(P xor (D and (S or P)))
88	D and S	A8	D and (P or S)
89	not(S xor (D or (P and (notS))))	A9	not(D xor (P or S))
8A	D and (S or (notP))	AA	D
8B	not(D xor (S or (P xor D)))	AB	D or (not(P or S))
8C	S and (D or (notP))	AC	S xor (P and (D xor S))
8D	not(S xor (D or (P xor S)))	AD	not(D xor (P or (S and D)))
8E	S xor ((S xor D) and (P xor D))	AE	D or (S and (notP))
8F	not(P and (not(D and S)))	AF	D or (notP)
90	P and (not(D xor S))	B0	P and (D or (notS))
91	not(S xor (D and (P or (notS))))	B1	not(P xor (D or (S xor P)))
92	D xor (P xor (S and (D or P)))	B2	S xor ((S xor P) or (D xor S))
93	not(S xor (P and D))	B3	not(S and (not(D and P)))

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
94	P xor (S xor (D and (P or S)))	B4	P xor (S and (notD))
95	not(D xor (P and S))	B5	not(D xor (P and (S or D)))
96	D xor (P xor S)	B6	D xor (P xor (S or (D and P)))
97	P xor (S xor (D or (not(P or S))))	B7	not(S and (D xor P))
98	not(S xor (D or (not(P or S))))	B8	P xor (S and (D xor P))
99	not(D xor S)	B9	not(D xor (S or (P and D)))
9A	D xor (P and (notS))	BA	D or (P and (notS))
9B	not(S xor (D and (P or S)))	BB	D or (notS)
9C	S xor (P and (notD))	BC	S xor (P and (not(D and S)))
9D	not(D xor (S and (P or D)))	BD	not((S xor D) and (P xor D))
9E	D xor (S xor (P or (D and S)))	BE	D or (P xor S)
9F	not(P and (D xor S))	BF	D or (not(P and S))

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Bit-Wise Operations and 8-bit Codes (C0 - FF)

Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
C0	P and S	E0	P and (D or S)
C1	not(S xor (P or (D and (notS))))	E1	not(P xor (D or S))
C2	not(S xor (P or (not(D or S))))	E2	D xor (S and (P xor D))
C3	not(P xor S)	E3	not(P xor (S or (D and P)))
C4	S and (P or (notD))	E4	S xor (D and (P xor S))
C5	not(S xor (P or (D xor S)))	E5	not(P xor (D or (S and P)))
C6	S xor (D and (notP))	E6	S xor (D and (not(P and S)))
C7	not(P xor (S and (D or P)))	E7	not((S xor P) and (P xor D))
C8	S and (D or P)	E8	S xor ((S xor P) and (D xor S))
C9	not(S xor (P or D))	E9	not(D xor (S xor (P and (not(D and S)))))
CA	D xor (P and (S xor D))	EA	D or (P and S)
CB	not(S xor (P or (D and S)))	EB	D or (not(P xor S))
CC	S	EC	S or (D and P)
CD	S or (not(D or P))	ED	S or (not(D xor P))
CE	S or (D and (notP))	EE	D or S
CF	S or (notP)	EF	S or (D or (notP))
D0	P and (S or (notD))	F0	P

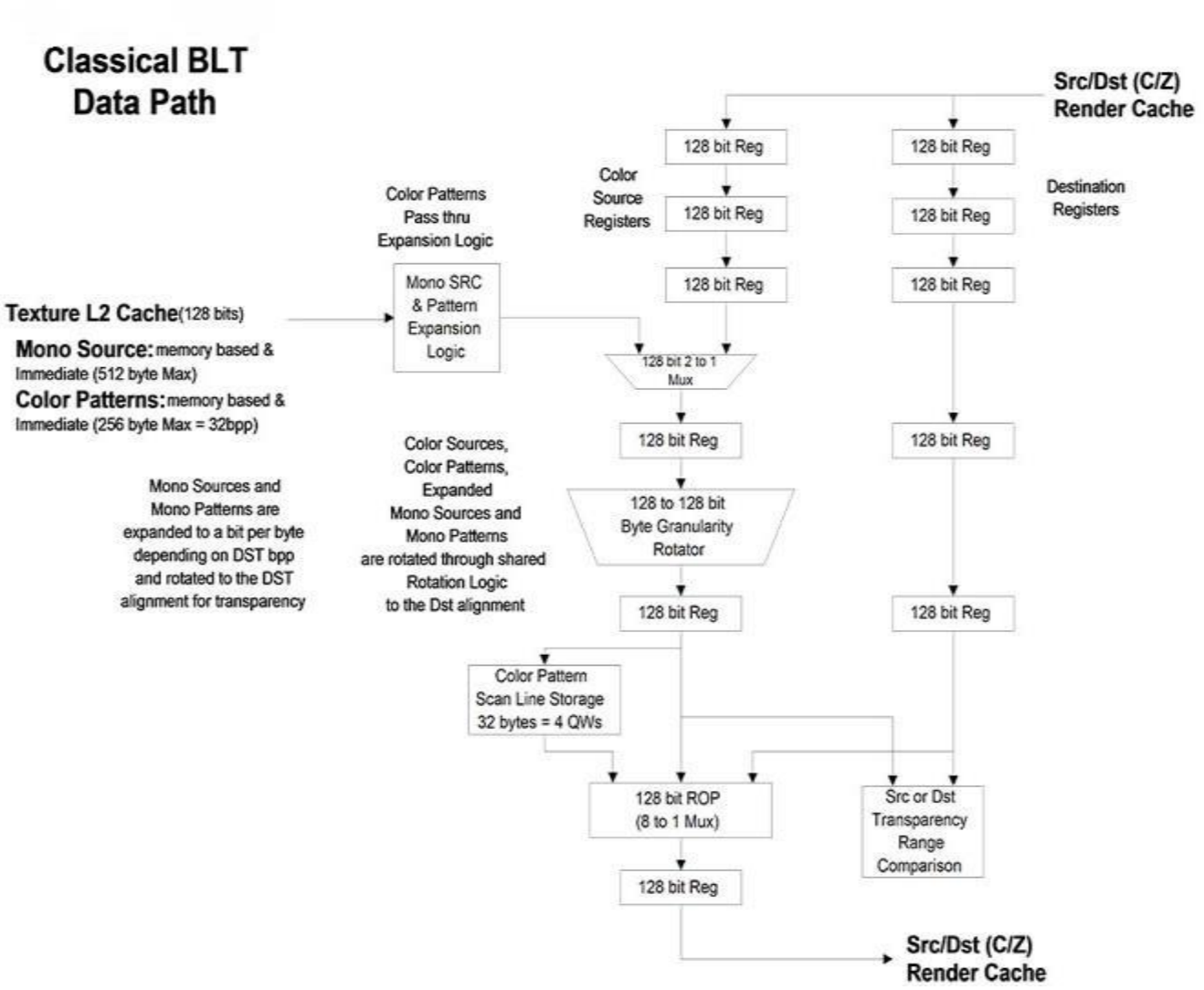
Code	Value Written to Bits at Destination	Code	Value Written to Bits at Destination
D1	not(P xor (S or (D xor P)))	F1	P or (not(D or S))
D2	P xor (D and (notS))	F2	P or (D and (notS))
D3	not(S xor (P and (D or S)))	F3	P or (notS)
D4	S xor ((S xor P) and (P xor D))	F4	P or (S and (notD))
D5	not(D and (not(P and S)))	F5	P or (notD)
D6	P xor (S xor (D or (P and S)))	F6	P or (D xor S)
D7	not(D and (P xor S))	F7	P or (not(D and S))
D8	P xor (D and (S xor P))	F8	P or (D and S)
D9	not(S xor (D or (P and S)))	F9	P or (not(D xor S))
DA	D xor (P and (not(S and D)))	FA	D or P
DB	not((S xor P) and (D xor S))	FB	D or (P or (notS))
DC	S or (P and (notD))	FC	P or S
DD	S or (notD)	FD	P or (S or (notD))
DE	S or (D xor P)	FE	D or (P or S)
DF	S or (not(D and P))	FF	writes all 1's

Notes:

S = Source Data
P = Pattern Data
D = Data Already Existing at the Destination

Per-Pixel Write-Masking Operations

The BLT engine is able to perform per-pixel write-masking with various data sources used as pixel masks to constrain which pixels at the destination are to be written to by the BLT engine. As shown in the figure below, either monochrome source or monochrome pattern data may be used as pixel masks. Color pattern data cannot be used. Another available pixel mask is derived by comparing a particular color range per color channel to either the color already specified for a given pixel at the destination or source.



Block Diagram and Data Paths of the BLT Engine

The command packets can specify the monochrome source or the monochrome pattern data as a pixel mask. When this feature is used, the bits that carry a value of 0 cause the bytes of the corresponding pixel at the destination to not be written to by the BLT engine, thereby preserving whatever data was originally carried within those bytes. This feature can be used in writing characters to the display, while also preserving the pre-existing backgrounds behind those characters. When both operands are in the transparent mode, the logical AND of the 2 operands are used for the write enables per pixel.

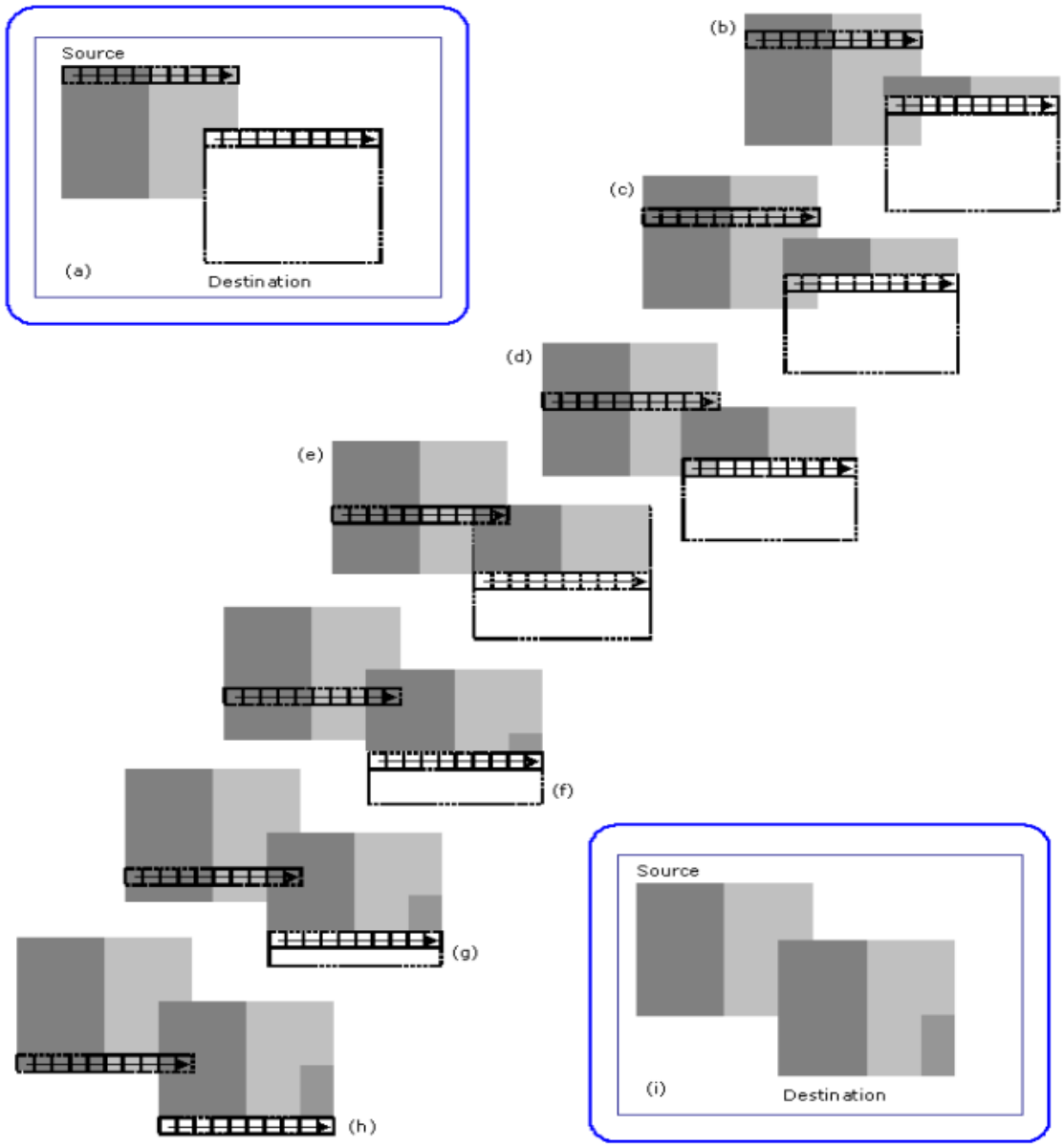
The 3-bit field, destination transparency mode, within the command packets can select per-pixel write-masking with a mask based on the results of color comparisons. The monochrome source background and foreground are range compared with either the bytes for the pixels at the destination or the source operand. This operation is described in the BLT command packet and register descriptions.

When the Source and Destination Locations Overlap

It is possible to have BLT operations in which the locations of the source and destination data overlap. This frequently occurs in BLT operations where a user is shifting the position of a graphical item on the display by only a few pixels. In these situations, the BLT engine must be programmed so that destination data is not written into destination locations that overlap with source locations before the source data at those locations has been read. Otherwise, the source data will become corrupted. The XY commands determine whether there is an overlap and perform the accesses in the proper direction to avoid data corruption.

The following figure shows how the source data can be corrupted when a rectangular block is copied from a source location to an overlapping destination location. The BLT engine typically reads from the source location and writes to the destination location starting with the left-most pixel in the top-most line of both, as shown in step (a). As shown in step (b), corruption of the source data has already started with the copying of the top-most line in step (a) — part of the source that originally contained lighter-colored pixels has now been overwritten with darker-colored pixels. More source data corruption occurs as steps (b) through (d) are performed. At step (e), another line of the source data is read, but the two right-most pixels of this line are in the region where the source and destination locations overlap, and where the source has already been overwritten as a result of the copying of the top-most line in step (a). Starting in step (f), darker-colored pixels can be seen in the destination where lighter-colored pixels should be. This errant effect occurs repeatedly throughout the remaining steps in this BLT operation. As more lines are copied from the source location to the destination location, it becomes clear that the end result is not what was originally intended.

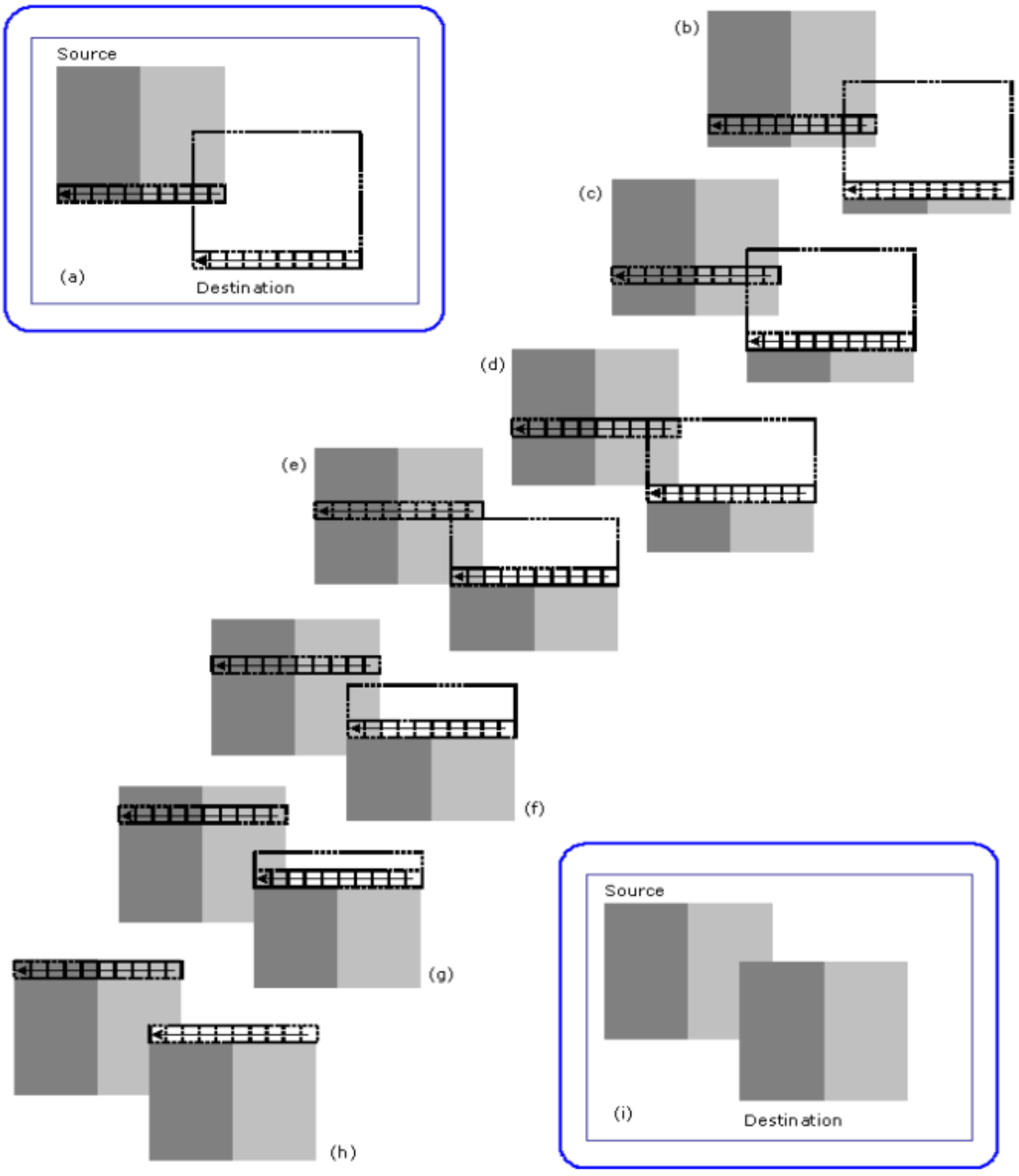
Source Corruption in BLT with Overlapping Source and Destination Locations



The BLT engine can alter the order in which source data is read and destination data is written when necessary to avoid source data corruption problems when the source and destination locations overlap. The command packets provide the ability to change the point at which the BLT engine begins reading and writing data from the upper left-hand corner (the usual starting point) to one of the other three corners. The BLT engine may be set to read data from the source and write it to the destination starting at any of the four corners of the panel.

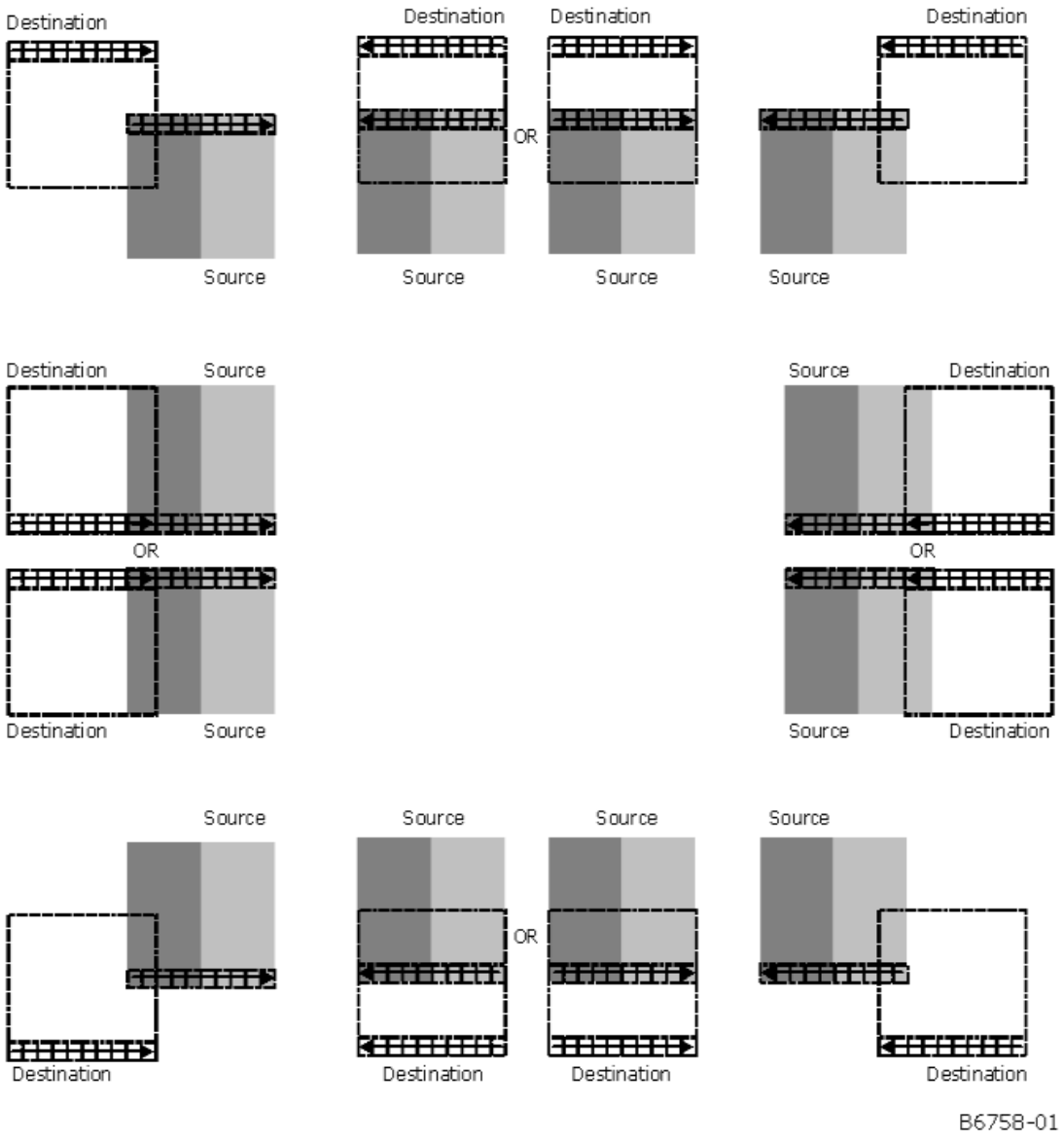
The XY command packets perform the necessary comparisons and start at the proper corner of each operand which avoids data corruption.

Correctly Performed BLT with Overlapping Source and Destination Locations



The following figure illustrates how this feature of the BLT engine can be used to perform the same BLT operation as was illustrated in the figure above, while avoiding the corruption of source data. As shown in the figure below, the BLT engine reads the source data and writes the data to the destination starting with the right-most pixel of the bottom-most line. By doing this, no pixel existing where the source and destination locations overlap will ever be written to before it is read from by the BLT engine. By the time the BLT operation has reached step (e) where two pixels existing where the source and destination locations overlap are about to be over written, the source data for those two pixels has already been read.

Suggested Starting Points for Possible Source and Destination Overlap Situations



The figure above shows the recommended lines and pixels to be used as starting points in each of 8 possible ways in which the source and destination locations may overlap. In general, the starting point should be within the area in which the source and destination overlap.

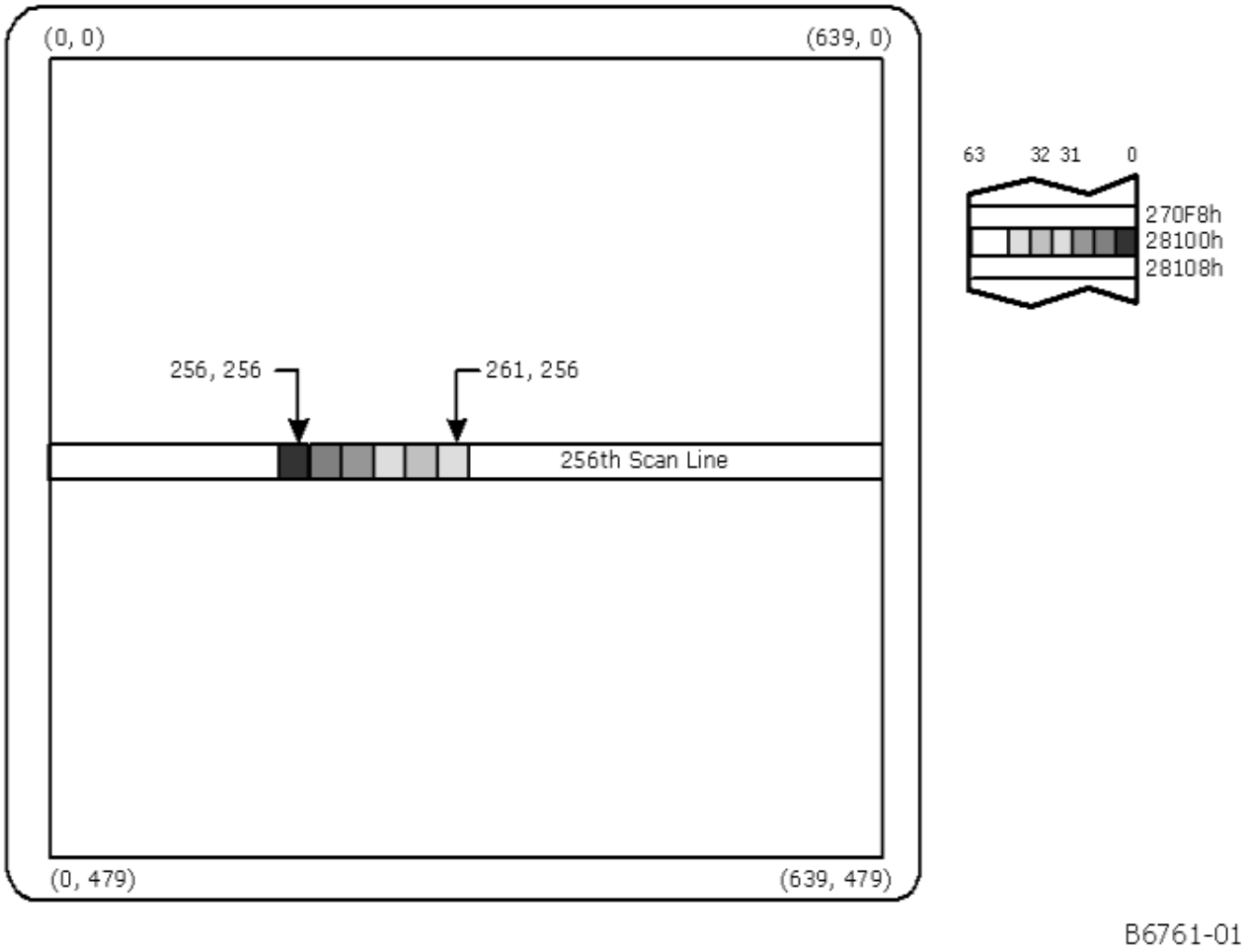
Basic Graphics Data Considerations

This topic is currently under development.

Contiguous vs. Discontinuous Graphics Data

Graphics data stored in memory, particularly in the frame buffer of a graphics system, has organizational characteristics that often distinguish it from other varieties of data. The main distinctive feature is the tendency for graphics data to be organized in a discontinuous block of graphics data made up of multiple sub-blocks of bytes, instead of a single contiguous block of bytes.

Representation of On-Screen Single 6-Pixel Line in the Frame Buffer

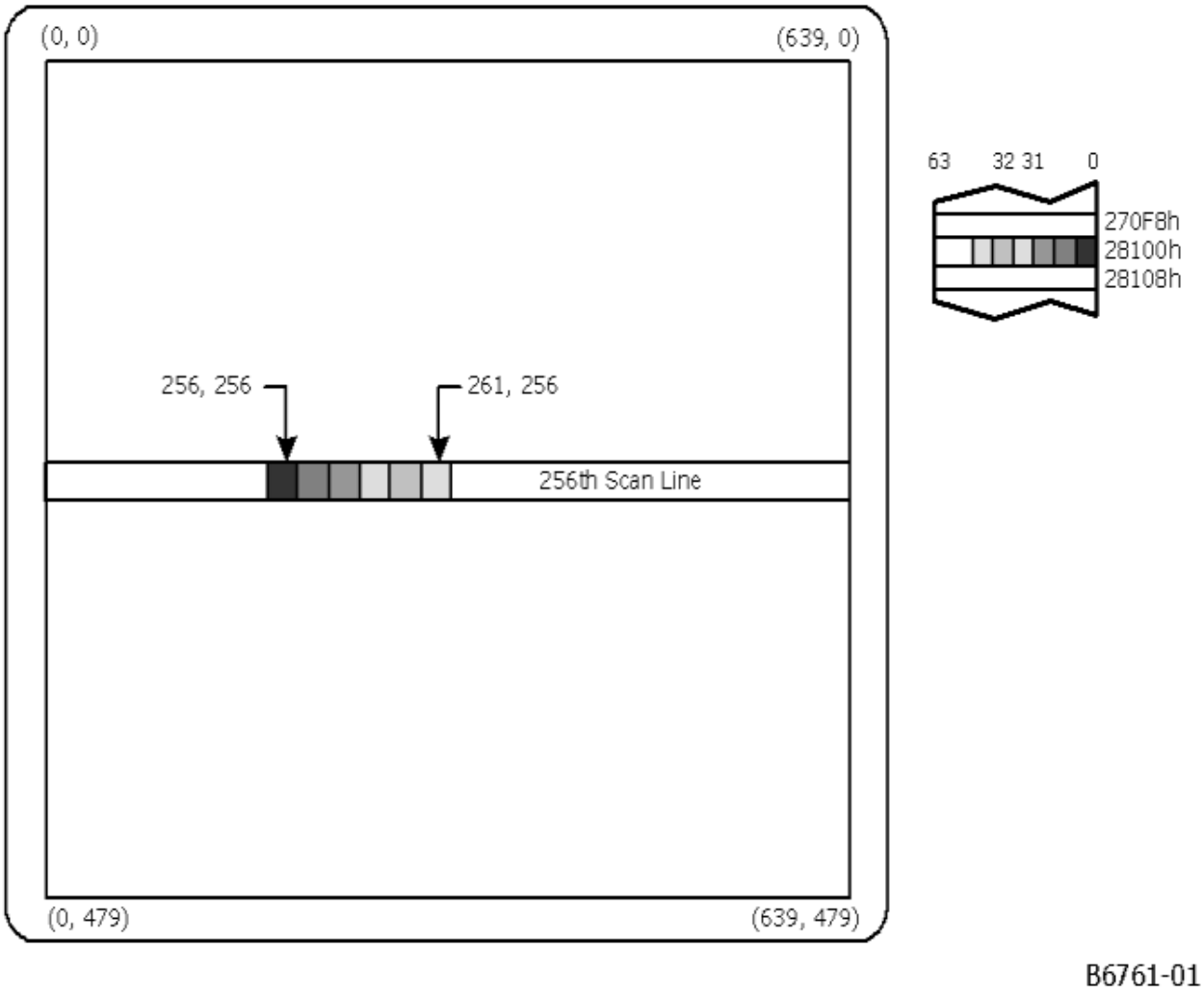


The figure above shows an example of contiguous graphics data — a horizontal line made up of six adjacent pixels within a single scan line on a display with a resolution of 640x480. Presuming that the graphics system driving this display has been set to 8 bits per pixel and that the frame buffer's starting address of 0h corresponds to the upper left-most pixel of this display, then the six pixels that make this horizontal line starting at coordinates (256, 256) occupies the six bytes starting at frame buffer address 28100h, and ending at address 28105h.

In this case, there is only one scan line's worth of graphics data in this single horizontal line, so the block of graphics data for all six of these pixels exists as a single, contiguous block comprised of only these six bytes. The starting address and the number of bytes are the only pieces of information that a BLT engine would require to read this block of data.

The simplicity of the above example of a single horizontal line contrasts sharply to the example of discontinuous graphics data depicted in the figure below. The simple six-pixel line of the figure above is now accompanied by three more six-pixel lines placed on subsequent scan lines, resulting in the 6x4 block of pixels shown.

Representation of On-Screen 6x4 Array of Pixels in the Frame Buffer



Since there are other pixels on each of the scan lines on which this 6x4 block exists that are not part of this 6x4 block, what appears to be a single 6x4 block of pixels on the display must be represented by a discontinuous block of graphics data made up of 4 separate sub-blocks of six bytes apiece in the frame buffer at addresses 28100h, 28380h, 28600h, and 28880h. This situation makes the task of reading what appears to be a simple 6x4 block of pixels more complex. However, there are two characteristics of this 6x4 block of pixels that help simplify the task of specifying the locations of all 24 bytes of this

discontinuous block of graphics data: all four of the sub-blocks are of the same length, and the four sub-blocks are separated from each other at equal intervals.

The BLT engine is designed to make use of these characteristics of graphics data to simplify the programming required to handle discontinuous blocks of graphics data. For such a situation, the BLT engine requires only four pieces of information: the starting address of the first sub-block, the length of a sub-block, the offset (in bytes), pitch, of the starting address of each subsequent sub-block, and the quantity of sub-blocks.

Source Data

The source data may exist in the frame buffer or elsewhere in the graphics aperture where the BLT engine may read it directly, or it may be provided to the BLT engine by the host CPU through the command packets. The block of source graphics data may be either contiguous or discontinuous, and may be either in color (with a color depth that matches that to which the BLT engine has been set) or monochrome.

The source select bit in the command packets specifies whether the source data exists in the frame buffer or is provided through the command packets. Monochrome source data is always specified as being supplied through an immediate command packet.

If the color source data resides within the frame buffer or elsewhere in the graphics aperture, then the Source Address Register, specified in the command packets is used to specify the address of the source.

In cases where the host CPU provides the source data, it does so by writing the source data to ring buffer directly after the BLT command that requires the data or uses an IMMEDIATE_INDIRECT_BLT command packet which has a size and pointer to the operand in Graphics aperture.

The block of bytes sent by the host CPU through the command packets must be quadword-aligned and the source data contained within the block of bytes must also be aligned.

To accommodate discontinuous source data, the source and destination pitch registers can be used to specify the offset in bytes from the beginning of one scan line's worth source data to the next. Otherwise, if the source data is contiguous, then an offset equal to the length of a scan line's worth of source data should be specified.

Monochrome Source Data

The opcode of the command packet specifies whether the source data is color or monochrome. Since monochrome graphics data only uses one bit per pixel, each byte of monochrome source data typically carries data for 8 pixels which hinders the use of byte-oriented parameters when specifying the location and size of valid source data. Some additional parameters must be specified to ensure the proper reading and use of monochrome source data by the BLT engine. The BLT engine also provides additional options for the manipulation of monochrome source data versus color source data.

The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome data, the BLT engine converts it into color through a process called color expansion, which takes place as a BLT operation is performed. In color expansion

the single bits of monochrome source data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry value corresponding to either the foreground or background color that have been specified for use in this conversion process. If a given bit of monochrome source data carries a value of 1, then the byte(s) of color data resulting from the conversion process will be set to carry the value of the foreground color. If a given bit of monochrome source data carries a value of 0, then the resulting byte(s) will be set to the value of the background color. The foreground and background colors used in the color expansion of monochrome source data can be set in the source expansion foreground color register and the source expansion background color register.

The BLT Engine requires that the bit alignment of each scan line's worth of monochrome source data be specified. Each scan line's worth of monochrome source data is word aligned but can actually start on any bit boundary of the first byte. Monochrome text is special cased and it is bit or byte packed, where in bit packed there are no invalid pixels (bits) between scan lines. There is a 3 bit field which indicates the starting pixel position within the first byte for each scan line, Mono Source Start.

Note that the Monosource surface start Base Address, should always be Cache Line (64byte) aligned.

The BLT engine also provides various clipping options for use with specific BLT commands (BLT_TEXT) with a monochrome source. Clipping is supported through: Clip rectangle Y addresses or coordinates and X coordinates along with scan line starting and ending addresses (with Y addresses) along with X starting and ending coordinates.

The maximum immediate source size is 128 bytes.

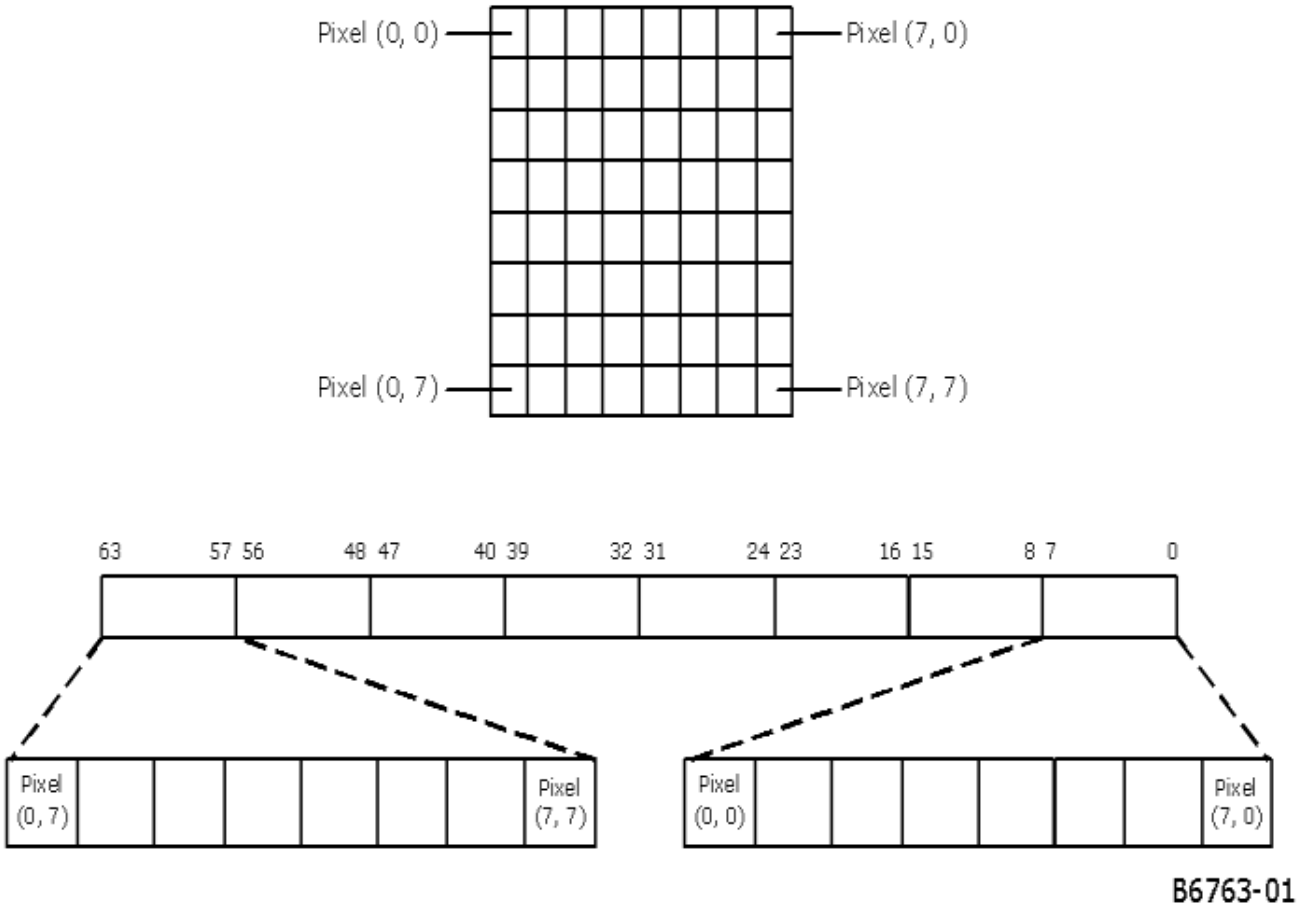
Pattern Data

The color pattern data must exist within the frame buffer or Graphics aperture where the BLT engine may read it directly or it can be sent through the command stream. The pattern data must be located in linear memory. Note also that the Color Pattern surface start Base Address, should always be Cache Line (64byte) aligned.

Monochrome pattern data is supplied by the command packet when it is to be used. As shown in figure below, the block of pattern graphics data always represents a block of 8x8 pixels. The bits or bytes of a block of pattern data may be organized in the frame buffer memory in only one of three ways, depending upon its color depth which may be 8, 16, or 32 bits per pixel (whichever matches the color depth to which the BLT engine has been set), or monochrome.

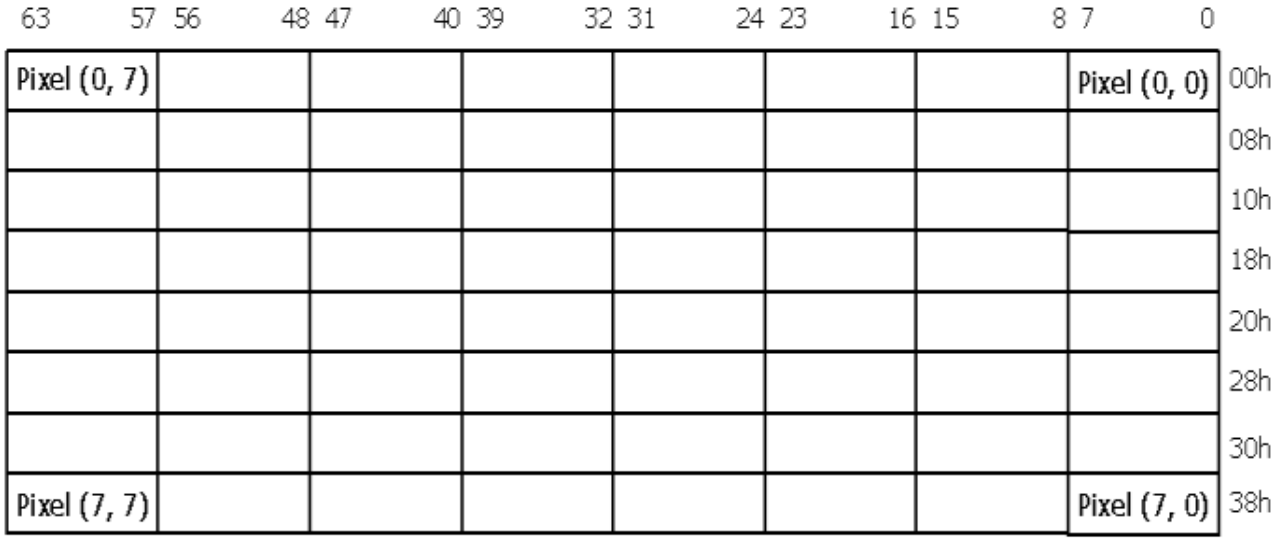
The maximum color pattern size is 256 bytes.

Pattern Data -- Always an 8x8 Array of Pixels



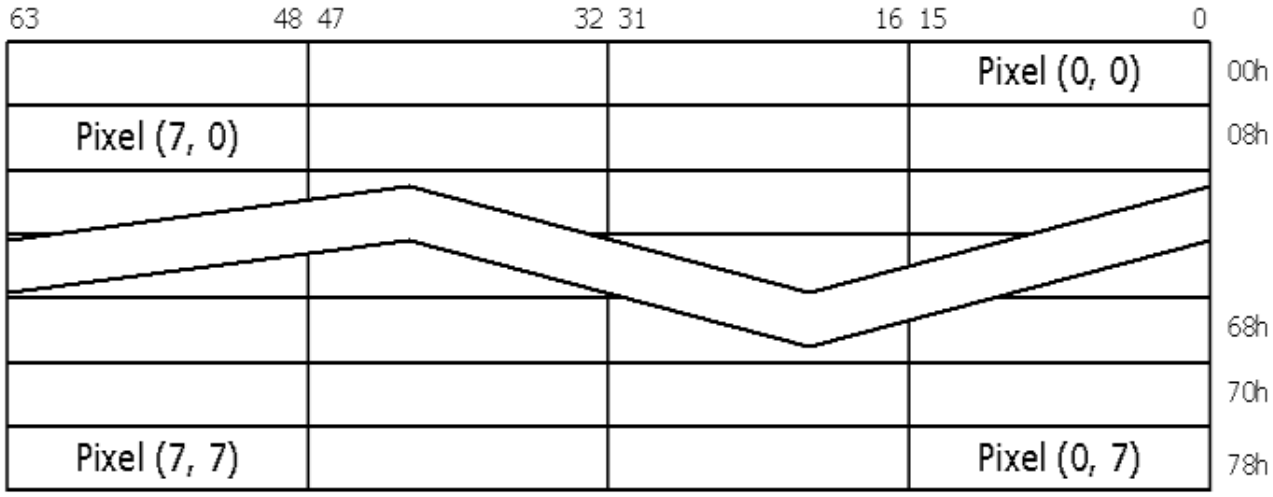
The Pattern Address Register is used to specify the address of the color pattern data at which the block of pattern data begins. The three least significant bits of the address written to this register are ignored, because the address must be in terms of quadwords. This is because the pattern must always be located on an address boundary equal to its size. Monochrome patterns take up 8 bytes, or a single quadword of space, and are loaded through the command packet that uses it. Similarly, color patterns with color depths of 8, 16, and 32 bits per pixel must start on 64-byte, 128-byte and 256-byte boundaries, respectively. The next 3 figures show how monochrome, 8bpp, 16bpp, and 32bpp pattern data , respectively, is organized in memory.

8bpp Pattern Data -- Occupies 64 Bytes (8 quadwords)



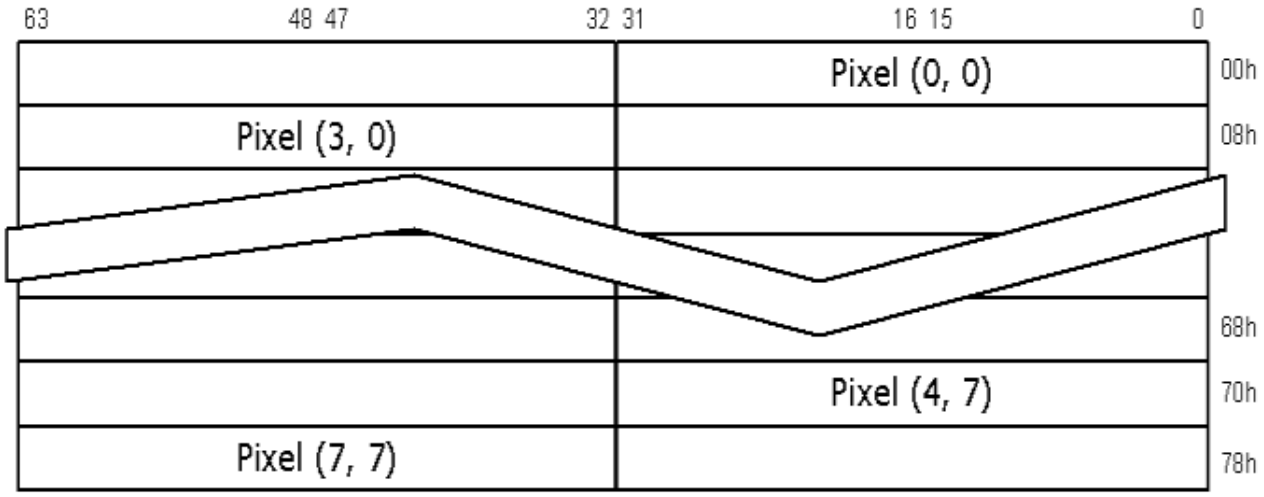
B6764-01

16bpp Pattern Data -- Occupies 128 Bytes (16 quadwords)



B6765-01

32bpp Pattern Data -- Occupies 256 Bytes (32 quadwords)



B6766-01

The opcode of the command packet specifies whether the pattern data is color or monochrome. The various bit-wise logical operations and per-pixel write-masking operations were designed to work with color data. In order to use monochrome pattern data, the BLT engine is designed to convert it into color through a process called “color expansion” which takes place as a BLT operation is performed. In color expansion, the single bits of monochrome pattern data are converted into one, two, or four bytes (depending on the color depth) of color data that are set to carry values corresponding to either the foreground or background color that have been specified for use in this process. The foreground color is used for pixels corresponding to a bit of monochrome pattern data that carry the value of 1, while the background color is used where the corresponding bit of monochrome pattern data carries the value of 0. The foreground and background colors used in the color expansion of monochrome pattern data can be set in the Pattern Expansion Foreground Color Register and Pattern Expansion Background Color Register.

Destination Data

There are actually two different types of “destination data”: the graphics data already residing at the location that is designated as the destination, and the data that is to be written into that very same location as a result of a BLT operation.

The location designated as the destination must be within the frame buffer or Graphics aperture where the BLT engine can read from it and write to it directly. The blocks of destination data to be read from and written to the destination may be either contiguous or discontinuous. All data written to the destination will have the color depth to which the BLT engine has been set. It is presumed that any data already existing at the destination which will be read by the BLT engine will also be of this same color depth — the BLT engine neither reads nor writes monochrome destination data.

The Destination Address Register is used to specify the address of the destination.

To accommodate discontinuous destination data, the Source and Destination Pitch Registers can be used to specify the offset in bytes from the beginning of one scan line’s worth of destination data to the next. Otherwise, if the destination data is contiguous, then an offset equal to the length of a scan line’s worth of destination data should be specified.

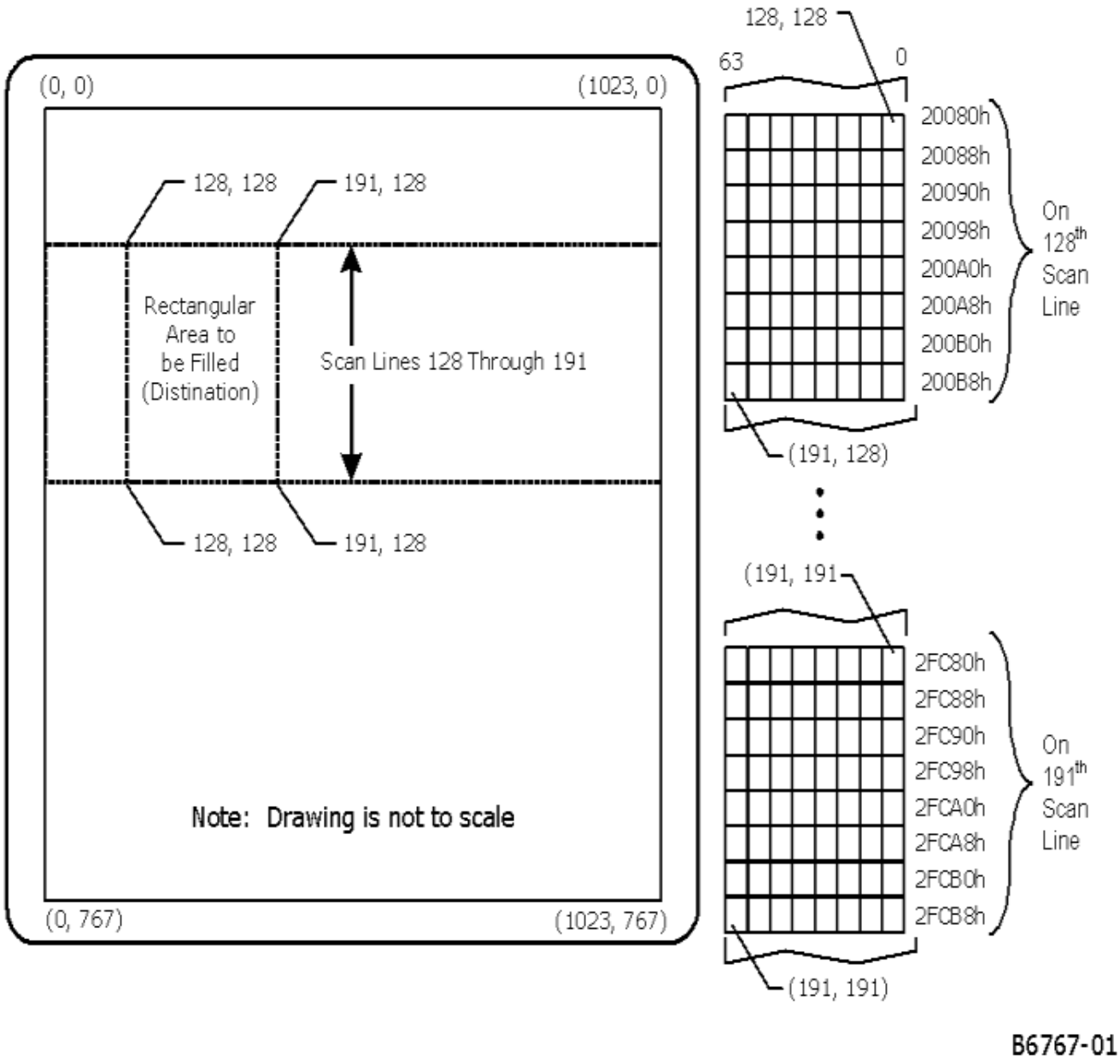
BLT Programming Examples

This topic is currently under development.

Pattern Fill — A Very Simple BLT

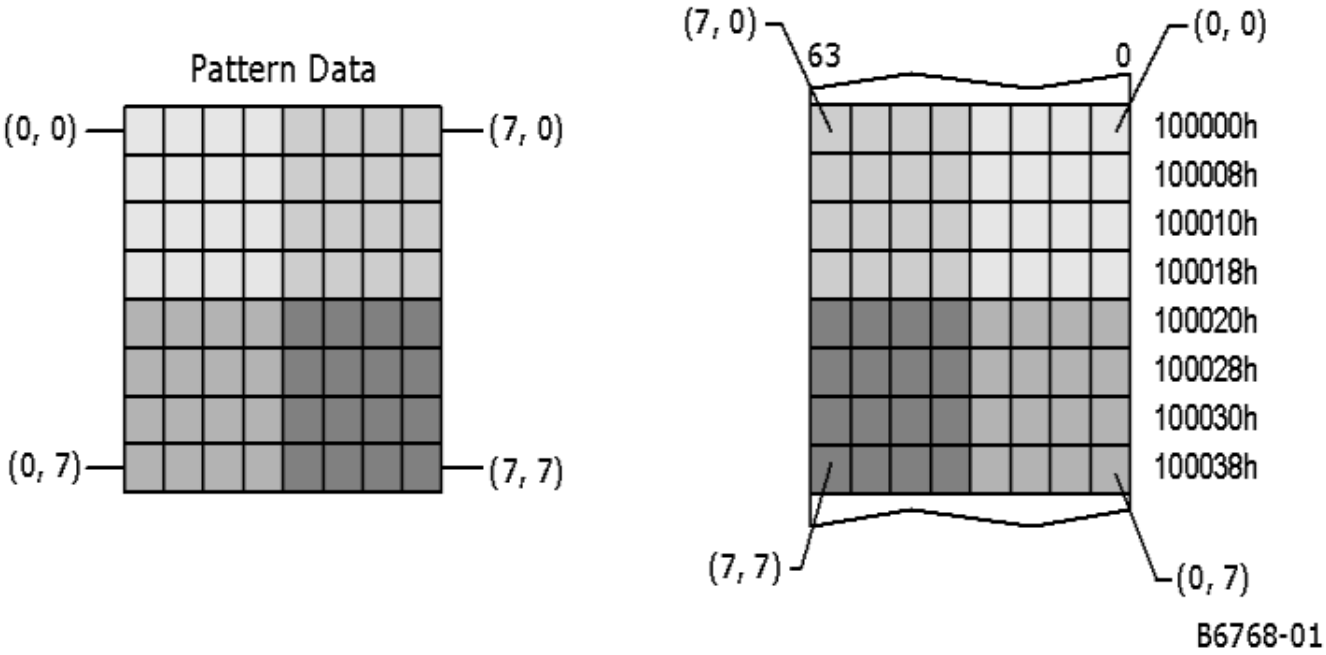
In this example, a rectangular area on the screen is to be filled with a color pattern stored as pattern data in off-screen memory. The screen has a resolution of 1024x768 and the graphics system has been set to a color depth of 8 bits per pixel.

On-Screen Destination for Example Pattern Fill BLT



As shown in the figure above, the rectangular area to be filled has its upper left-hand corner at coordinates (128, 128) and its lower right-hand corner at coordinates (191, 191). These coordinates define a rectangle covering 64 scan lines, each scan line's worth of which is 64 pixels in length — in other words, an array of 64x64 pixels. Presuming that the pixel at coordinates (0, 0) corresponds to the byte at address 00h in the frame buffer memory, the pixel at (128, 128) corresponds to the byte at address 20080h.

Pattern Data for Example Pattern Fill BLT



As shown in figure above, the pattern data occupies 64 bytes starting at address 100000h. As always, the pattern data represents an 8x8 array of pixels.

The BLT command packet is used to select the features to be used in this BLT operation, and must be programmed carefully. The vertical alignment field should be set to 0 to select the top-most horizontal row of the pattern as the starting row used in drawing the pattern starting with the top-most scan line covered by the destination. The pattern data is in color with a color depth of 8 bits per pixel, so the dynamic color enable should be asserted with the dynamic color depth field should be set to 0. Since this BLT operation does not use per-pixel write-masking (destination transparency mode), this field should be set to 0. Finally, the raster operation field should be programmed with the 8-bit value of F0h to select the bit-wise logical operation in which a simple copy of the pattern data to the destination takes place. Selecting this bit-wise operation in which no source data is used as an input causes the BLT engine to automatically forego either reading source data from the frame buffer.

The Destination Pitch Register must be programmed with number of bytes in the interval from the start of one scan line's worth of destination data to the next. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024.

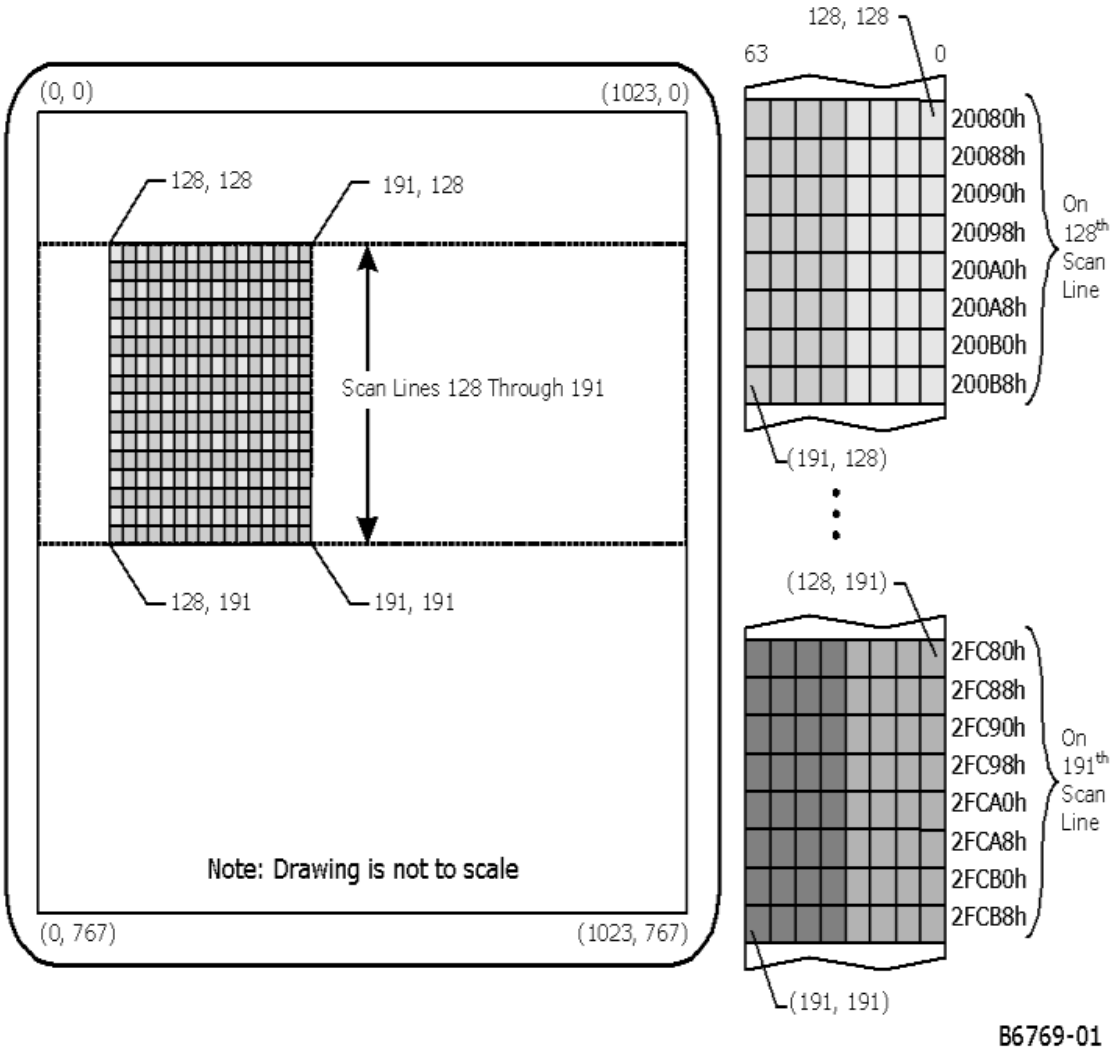
Bits [31:3] of the Pattern Address Register must be programmed with the address of the pattern data.

Similarly, bits [31:0] of the Destination Address Register must be programmed with the byte address at the destination that will be written to first. In this case, the address is 20080h, which corresponds to the byte representing the pixel at coordinates (128, 128).

This BLT operation does not use the values in the Source Address Register or the Source Expansion Background or Foreground Color Registers.

The Destination Width and Height Registers (or the Destination X and Y Coordinates) must be programmed with values that describe to the BLT engine the 64x64 pixel size of the destination location. The height should be set to carry the value of 40h, indicating that the destination location covers 64 scan lines. The width should be set to carry the value of 40h, indicating that each scan line's worth of destination data occupies 64 bytes. All of this information is written to the ring buffer using the PAT_BLT (or XY_PAT_BLT) command packet.

Results of Example Pattern Fill BLT

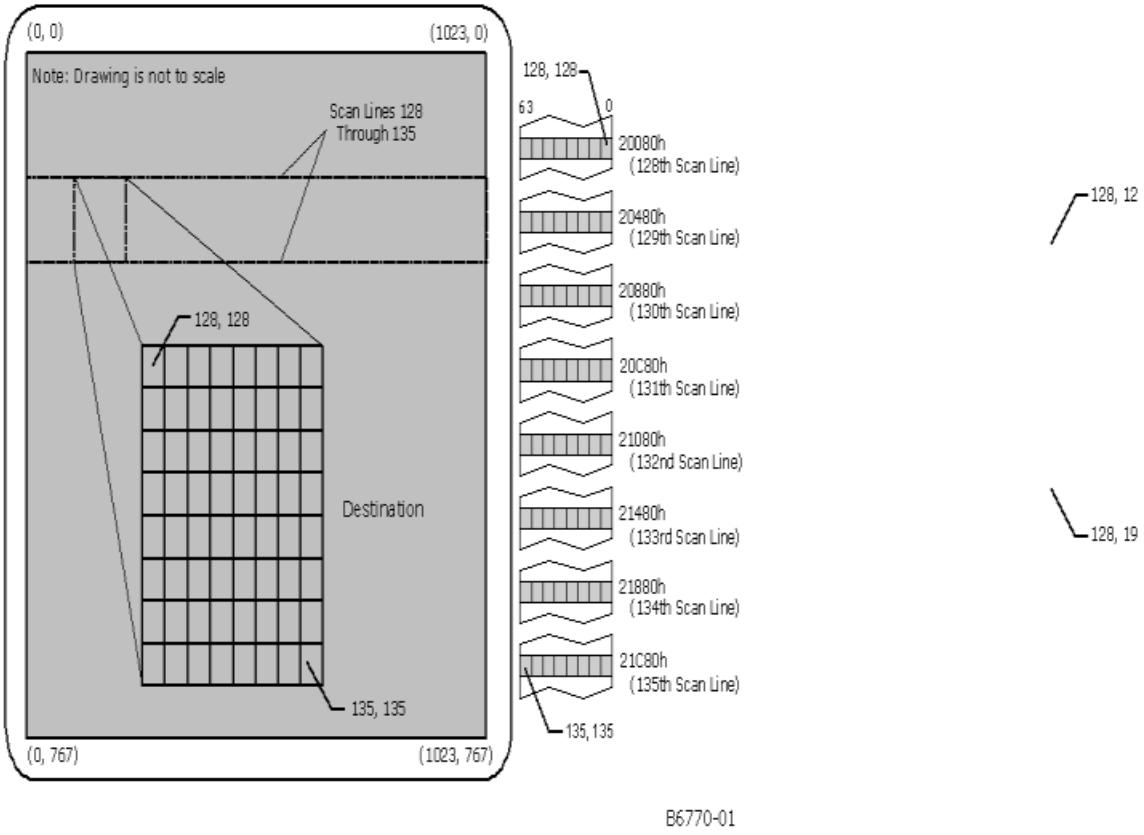


The figure above shows the end result of performing this BLT operation. The 8x8 pattern has been repeatedly copied ("tiled") into the entire 64x64 area at the destination.

Drawing Characters Using a Font Stored in System Memory

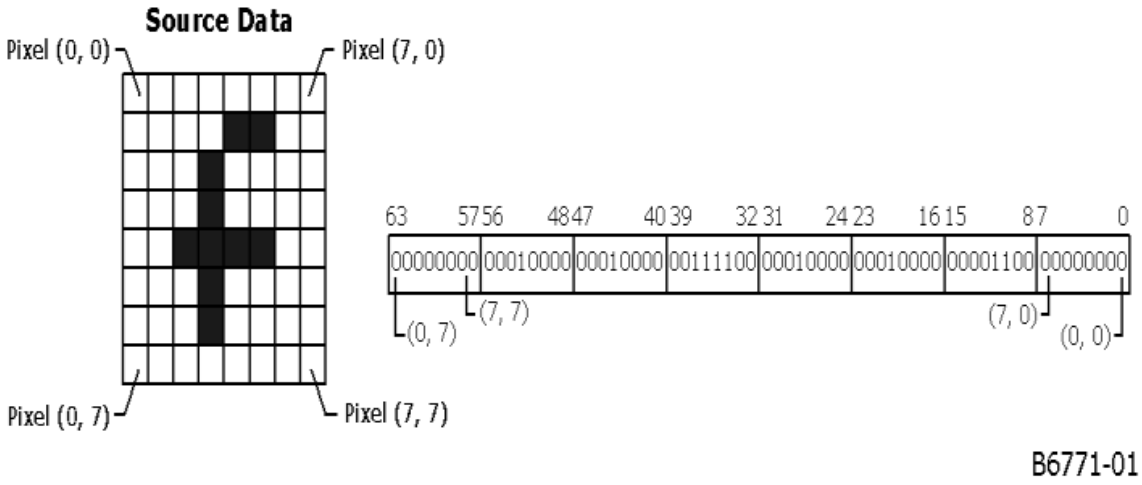
In this example BLT operation, a lowercase letter “f” is to be drawn in black on a display with a gray background. The resolution of the display is 1024x768, and the graphics system has been set to a color depth of 8 bits per pixel.

On-Screen Destination for Example Character Drawing BLT



The figure above shows the display on which this letter “f” is to be drawn. As shown in this figure, the entire display has been filled with a gray color. The letter “f” is to be drawn into an 8x8 region on the display with the upper left-hand corner at the coordinates (128, 128).

Source Data in System Memory for Example Character Drawing BLT



The figure above shows both the 8x8 pattern making up the letter “f” and how it is represented somewhere in the host’s system memory — the actual address in system memory is not important. The letter “f” is represented in system memory by a block of monochrome graphics data that occupies 8 bytes. Each byte carries the 8 bits needed to represent the 8 pixels in each scan line’s worth of this graphics data. This type of pattern is often used to store character fonts in system memory.

During this BLT operation, the host CPU will read this representation of the letter “f” from system memory, and write it to the BLT engine by performing memory writes to the ring buffer as an immediate monochrome BLT operand following the BLT_TEXT command. The BLT engine will receive this data through the command stream and use it as the source data for this BLT operation. The BLT engine will be set to the same color depth as the graphics system — 8 bits per pixel, in this case. Since the source data in this BLT operation is monochrome, color expansion must be used to convert it to an 8 bpp color depth. To ensure that the gray background behind this letter “f” is preserved, per-pixel write masking will be performed, using the monochrome source data as the pixel mask.

The BLT Setup and Text_immediate command packets are used to select the features to be used in this BLT operation. Only the fields required by these two command packets must be programmed carefully. The BLT engine ignores all other registers and fields. The source select field in the Text_immediate command must be set to 1, to indicate that the source data is provided by the host CPU through the command packet. Finally, the raster operation field should be programmed with the 8-bit value CCh to select the bit-wise logical operation that simply copies the source data to the destination. Selecting this bit-wise operation in which no pattern data is used as an input, causes the BLT engine to automatically forego reading pattern data from the frame buffer.

The Setup Pattern/Source Expansion Foreground Color Register to specify the color with which the letter “f” will be drawn. There is no Source address. All scan lines of the glyph are bit packed and the clipping is controlled by the ClipRect registers from the SETUP_BLT command and the Destination Y1, Y2, X1, and X2 registers in the TEXT_BLT command. Only the pixels that are within (inclusive comparisons) the clip rectangle are written to the destination surface.

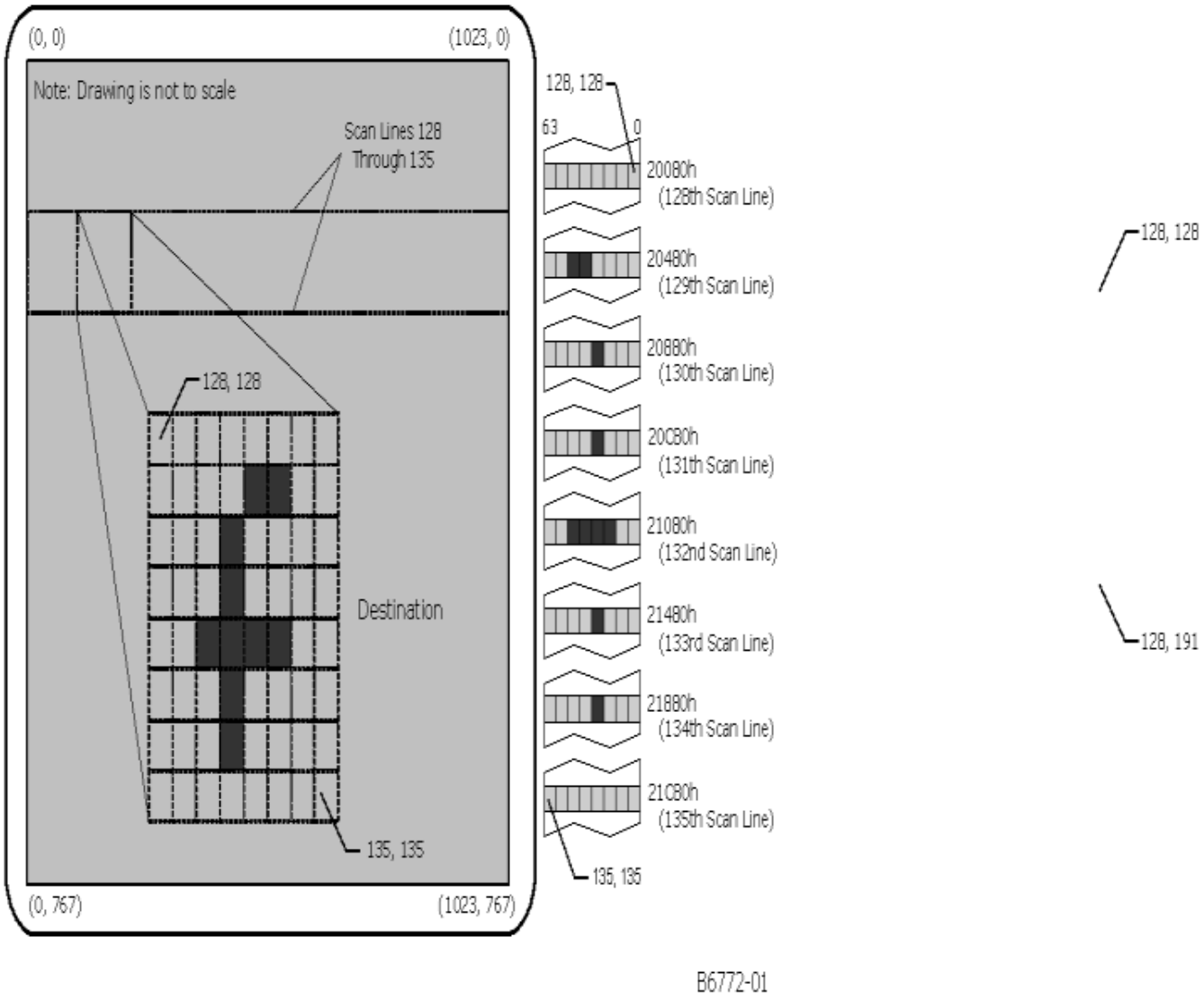
The Destination Pitch Register must be programmed with a value equal to the number of bytes in the interval between the first bytes of each adjacent scan line's worth of destination data. Since the color depth is 8 bits per pixel and the horizontal resolution of the display is 1024 pixels, the value to be programmed into these bits is 400h, which is equal to the decimal value of 1024. Since the source data used in this BLT operation is monochrome, the BLT engine will not use a byte-oriented pitch value for the source data.

Since the source data is monochrome, color expansion is required to convert it to color with a color depth of 8 bits per pixel. Since the Setup Pattern/Source Expansion Foreground Color Register is selected to specify the foreground color of black to be used in drawing the letter "f", this register must be programmed with the value for that color. With the graphics system set for a color depth of 8 bits per pixel, the actual colors are specified in the RAMDAC palette, and the 8 bits stored in the frame buffer for each pixel actually specify the index used to select a color from that palette. This example assumes that the color specified at index 00h in the palette is black, and therefore bits [7:0] of this register should be set to 00h to select black as the foreground color. The BLT engine ignores bits [31:8] of this register because the selected color depth is 8 bits per pixel. Even though the color expansion being performed on the source data normally requires that both the foreground and background colors be specified, the value used to specify the background color is not important in this example. Per-pixel write-masking is being performed with the monochrome source data as the pixel mask, which means that none of the pixels in the source data that will be converted to the background color will ever be written to the destination. Since these pixels will never be seen, the value programmed into the Pattern/Source Expansion Background Color Register to specify a background color is not important.

The Destination Width and Height Registers are not used. The Y1, Y2, X1, and X2 are used to describe to the BLT engine the 8x8 pixel size of the destination location. The Destination Y1 and Y2 address (or coordinate) registers must be programmed with the starting and ending scan line address (or Y coordinates) of the destination data. This address is specified as an offset from the start of the frame buffer of the scan line at the destination that will be written to first. The destination X1 and X2 registers must be programmed with the starting and ending pixel offsets from the beginning of the scan line.

This BLT operation does not use the values in the Pattern Address Register, the Source Expansion Background Color Register, or the Source Expansion Foreground Color Register.

Results of Example Character Drawing BLT



The preceding shows the end result of performing this BLT operation. Only the pixels that form part of the actual letter "f" have been drawn into the 8x8 destination location on the display, leaving the other pixels within the destination with their original gray color.

BLT Instruction Overview

This chapter defines the instructions used to control the 2D (BLT) rendering function.

The instructions detailed in this chapter are used across devices. However, slight changes may be present in some instructions (i.e., for features added or removed), or some instructions may be removed entirely. Refer to the *Device Dependencies* chapter for summary information regarding device-specific behaviors/interfaces/features.

The XY instructions offload the drivers by providing X and Y coordinates and taking care of the access directions for overlapping BLTs without fields specified by the driver.

Color pixel sizes supported are 8, 16, and 32 bits per pixel (bpp). All pixels are naturally aligned.

BLT Engine State

Most of the BLT instructions are state-free, which means that all states required to execute the command is within the instruction. If clipping is not used, then there is no shared state for many of the BLT instructions. This allows the BLT Engine to be shared by many drivers with minimal synchronization between the drivers.

Instructions which share state are:

All instructions that are X,Y commands and use the Clipping Rectangle by asserting the Clip Enable field

All XY_Setup Commands (XY_SETUP_BLT and XY_SETUP_MONO_PATTERN_SL_BLT, XY_SETUP_CLIP_BLT) load the shared state for the following commands:

XY_PIXEL_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_SCANLINES_BLT	
XY_TEXT_BLT	(Negative Stride (=Pitch) Not Allowed)
XY_TEXT_IMMEDIATE_BLT	(Negative Stride (=Pitch) Not Allowed)

State registers that are saved & restored in the Logical Context:

BR1+	Setup Control (Solid Pattern Select, Clipping Enable, Mono Source Transparency Mode, Mono Pattern Transparency Mode, Color Depth[1:0], Raster Operation[7:0], & Destination Pitch[15:0]) + 32bpp Channel Mask[1:0], Mono / Color Pattern
BR05	Setup Background Color
BR06	Setup Foreground Color
BR07	Setup Pattern Base Address
BR09	Setup Destination Base Address
BR20	DW0 for a Monochrome Pattern
BR21	DW1 for a Monochrome Pattern
BR24	ClipRectY1’X1
BR25	ClipRectY2’X2

Cacheable Memory Support

The BLT Engine can be used to transfer data between cacheable ("system") memory and uncached ("main", or "UC") graphics memory using the BLT instructions. The GTT must be properly programmed to map memory pages as cacheable or UC. Only linear-mapped (not tiled) surfaces can be mapped as cacheable.

Transfers between cacheable sources and cacheable destinations are not supported. Patterns and monochrome sources can not be located in cacheable memory.

Cacheable write operands do not snoop the processor's cache nor update memory until evicted from the render cache. Cacheable read or write operands are not snooped (nor invalidated) from either internal cache by external (processor, hublink,...) accesses.

Device Cache Coherency: Render & Texture Caches

Software must initiate cache flushes to enforce coherency between the render and texture caches, i.e., both the render and texture caches must be flushed before a BLT destination surface can be reused as a texture source. Color sources and destinations use the render cache, while patterns and monochrome sources use the texture cache.

BLT Engine Instructions

The Instruction Target field is used as an opcode by the BLT Engine state machine to qualify the control bits that are relevant for executing the instruction. The descriptions for each DWord and bit field are contained in the *BLT Engine Instruction Field Definition* section. Each DWord field is described as a register, but none of these registers can be written or read through a memory mapped location; they are internal state only.

BLT Programming Restrictions

Overlapping Source/Destination BLTs:

For all products *negative pitch* programming is allowed only when the source and destination surfaces are of the same type: linear source to linear destination copy, or tiled source to tiled destination copy. This is a must requirement. In such cases:

- Both the pitches must be programmed to be a negative value, if the source and destination surfaces are overlapping.
- Either of the pitches can be programmed to a negative value if required to do mirroring, but only if the complete source and destination surfaces are not overlapping anywhere on the surfaces.

Legacy Blits:

The following condition must be avoided when programming the BLT engine: Linear surfaces with a cache line in scan line Y for the source stream overlapping with a cache line in scan line Y-1 for the dest stream (=> non-aligned surface pitches). The cache coherency rules combined with the Blitter data consumption rules result in UNDEFINED operation.

All reserved fields must be programmed to 0s.

When using monosource or text data (bit/byte/word aligned): Do not program pixel widths greater than 32,745 pixels.

The other way to do this is driver should always program a dummy 3D.

NON-PIPELINE state following the BLT commands:

Description
For Monosource and Color Pattern surfaces, and also linear colour source and destination surfaces, the start **Base Address** programmed should always be Cache Line (64 byte) aligned.

2D (XY) BLT Instructions

Most BLT instructions (prefixed with "XY_") use 2D X,Y coordinate specifications vs. lower-level linear addresses These instructions also support simple 2D clipping against a clip rectangle. The top and left Clipping coordinates are inclusive. The bottom and right coordinates are exclusive. The BLT Engine performs a trivial reject for all CLIP BLT instructions before performing any accesses.

Negative destination and source coordinates are supported. In the case of negative source coordinates, the destination X1 and Y1 are modified by the absolute value of the negative source coordinate before the destination clip checking and final drawing coordinates are calculated. The absolute value of the source negative coordinate is added to the corresponding destination coordinate. The BLT engine clipping also checks for (DX2 [or = DX1) or (DY2 [or = DY1) after this calculation and if true, then the BLT is totally rejected.

Source and destination pitches have the additional explanation given next. The below statements are applicable for pitch field in all of the Blit commands:

1. For Linear surfaces, the pitch is programmed in bytes. For Tiled surfaces the pitch programmed is in Dwords count.
2. For Tiled surfaces this pitch is of 512Byte granularity for Tile-X: This means the tiled-x surface pitch can be (512, 1024, 1536, 2048...)/4 (in Dwords).
3. For Tiled surfaces this pitch is of 128B granularity for Tile-Y: This means the tiled-y surface pitch can be (128, 256, 384, 512...)/4 (in Dwords).
4. Another way to indicate this is, for tiled surfaces, the pitch is programmed in Dwords and is an integral multiple of the tile width.

XY_COLOR_BLT

XY_FULL_BLT

XY_FULL_IMMEDIATE_PATTERN_BLT

XY_FULL_MONO_PATTERN_BLT

XY_FULL_MONO_PATTERN_MONO_SRC_BLT

XY_FULL_MONO_SRC_BLT

XY_FULL_MONO_SRC_IMMEDIATE_PATTERN_BLT

XY_MONO_PAT_BLT

XY_MONO_PAT_FIXED_BLT

XY_MONO_SRC_COPY_BLT

XY_MONO_SRC_COPY_IMMEDIATE_BLT

XY_PAT_BLT

XY_PAT_BLT_IMMEDIATE

XY_PAT_CHROMA_BLT

XY_PAT_CHROMA_BLT_IMMEDIATE

XY_PIXEL_BLT

XY_SCANLINES_BLT

XY_SETUP_BLT

XY_SETUP_CLIP_BLT

XY_SETUP_MONO_PATTERN_SL_BLT

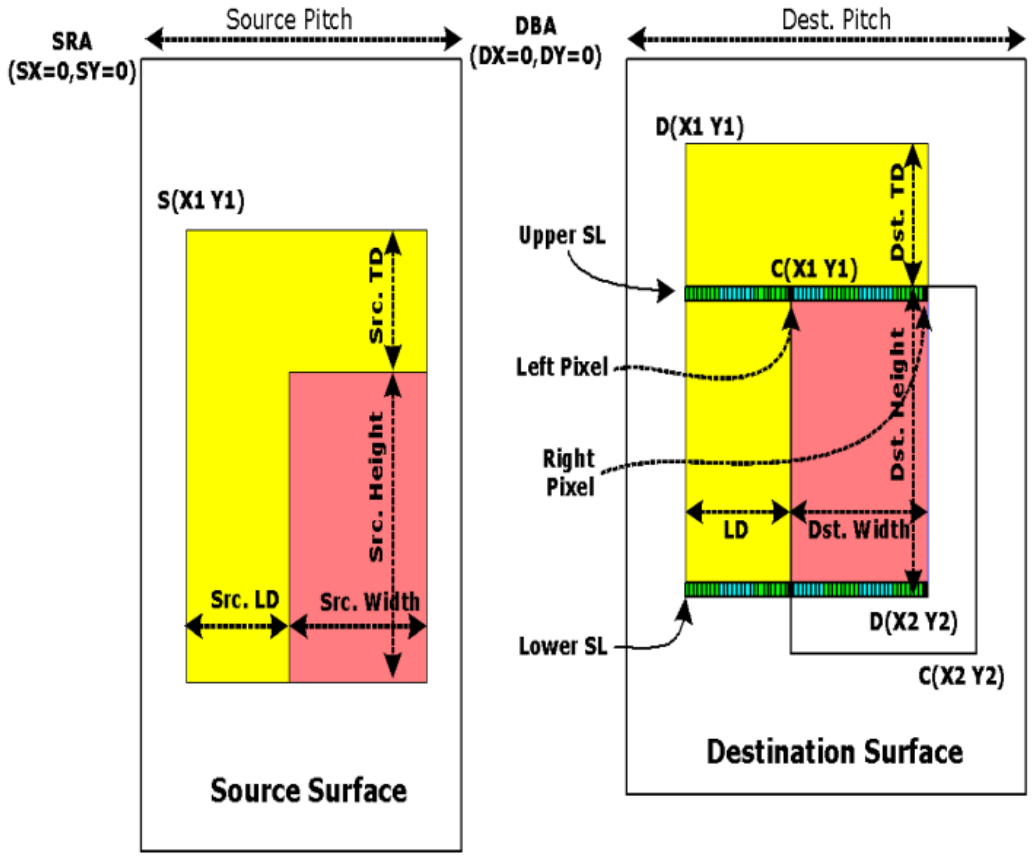
XY_SRC_COPY_BLT

XY_SRC_COPY_CHROMA_BLT

XY_TEXT_BLT

XY_TEXT_IMMEDIATE_BLT

Some Equalities & Inequalities for Source Clipping



Some Equalities & Inequalities for Source Clipping:

Src. TD = Dst. TD (Top discard in SL)
Src LD = LD (Left Discard in Pixels)
Src Height = Dst. Height in SL
Src Width = Dst. Width in Pixels

Note: Src. Pitch is not equal to Dst. Pitch

B6773-01

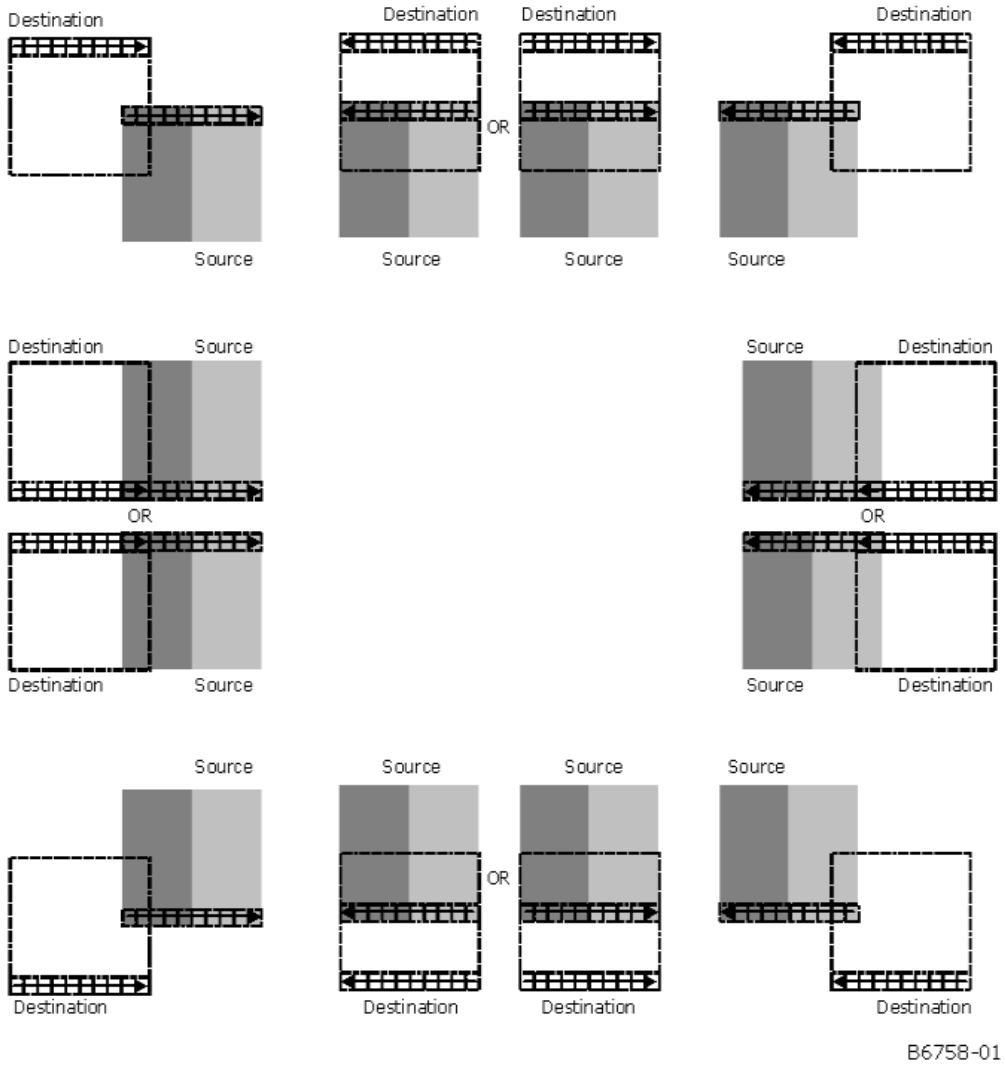
DX1, DY1, CX1, and CY1 are inclusive, while DX2, DY2, CX2, and CY2 are exclusive.

Destination pixel address = (Destination Base Address + (Destination Y coordinate * Destination pitch) + (Destination X coordinate * bytes per pixel)).

Source pixel address = (Source Base Address + (Source Y coordinate * Source pitch) + (Source X coordinate * bytes per pixel)).

Since there is 1 set of Clip Rectangle registers, the Interrupt Ring BLT commands either MUST NEVER enable clipping with these command and never use the XY_Pixel_BLT, XY_Scanline_BLT, nor XY_Text_BLT commands or it must use context switching. The Interrupt rings can also use the non-clipped, linear address commands specified before this section.

The base addresses plus the X and Y coordinates determine if there is an overlap between the source and destination operands. If the base addresses of the source and destination are the same and the Source X1 is less than Destination X1, then the BLT Engine performs the accesses in the X-backwards access pattern. There is no need to look for an actual overlap. If the base addresses are the same and Source Y1 is less than Destination Y1, then the scan line accesses are performed backwards.



BLT Engine Instruction Field Definitions

This section describes the BLT Engine instruction fields. These descriptions are in the format of register descriptions. These registers are internal and are not readable. Some of these registers are state that is saved and restored for supporting separate software threads.

BR00 - BLT Opcode and Control

BR01 - Setup BLT Raster OP, Control, and Destination Offset

BR05 - Setup Expansion Background Color

BR06 - Setup Expansion Foreground Color

BR07 - Setup Blit Color Pattern Address Lower Order Address bits

BR30 - Setup Blit Color Pattern Address Higher Order Address

BR09 - Destination Address Lower Order Address Bits

BR27 - Destination Address Higher Order Address

BR11 - BLT Source Pitch (Offset)

BR12 - Source Address Lower order Address bits

BR28 - Source Address Higher order Address

BR13 - BLT Raster OP, Control, and Destination Pitch

BR14 - Destination Width and Height

BR15 - Color Pattern Address Lower order Address bits

BR29 - Color Pattern Address Higher order Address

BR16 - Pattern Expansion Background and Solid Pattern Color

BR17 - Pattern Expansion Foreground Color

BR18 - Source Expansion Background and Destination Color

BR19 - Source Expansion Foreground Color

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 12: Display

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

VGA and Extended VGA Registers 1
General Control and Status Registers 2
ST00 - Input Status 0 3
ST01 - Input Status 1 4
FCR - Feature Control 6
MSR - Miscellaneous Output 7
Sequencer Registers 9
SRX - Sequencer Index 9
SR00 - Sequencer Reset 9
SR01 - Clocking Mode 10
SR02 - Plane/Map Mask 12
SR03 - Character Font 13
SR04 - Memory Mode Register 15
SR07 - Horizontal Character Counter Reset 16
Graphics Controller Registers 17
GRX - GRX Graphics Controller Index Register 17
GR00 - Set/Reset Register 17
GR01 - Enable Set/Reset Register 18
GR02 - Color Compare Register 18
GR03 - Data Rotate Register 19
GR04 - Read Plane Select Register 20
GR05 - Graphics Mode Register 21
GR06 - Miscellaneous Register 25
GR07 - Color Don't Care Register 26
GR08 - Bit Mask Register 26
Attribute Controller Registers 27
ARX - Attribute Controller Index Register 27
AR[00:0F] - Palette Registers [0:F] 28
AR10 - Mode Control Register 29
AR11 - Overscan Color Register 31
AR12 - Memory Plane Enable Register 32
AR13 - Horizontal Pixel Panning Register 33

AR14 - Color Select Register....34
VGA Color Palette Registers....35
DACMASK - Pixel Data Mask Register....36
DACSTATE - DAC State Register....36
DACRX - Palette Read Index Register....37
DACWX - Palette Write Index Register....37
DACDATA - Palette Data Register....38
CRT Controller Register....39
CRX - CRT Controller Index Register....40
CR00 - Horizontal Total Register....40
CR01 - Horizontal Display Enable End Register....41
CR02 - Horizontal Blanking Start Register....41
CR03 - Horizontal Blanking End Register....42
CR04 - Horizontal Sync Start Register....43
CR05 - Horizontal Sync End Register....44
CR06 - Vertical Total Register....45
CR07 - Overflow Register (Vertical)....46
CR08 - Preset Row Scan Register....48
CR09 - Maximum Scan Line Register....49
CR0A - Text Cursor Start Register....50
CR0B - Text Cursor End Register....51
CR0C - Start Address High Register....51
CR0D - Start Address Low Register....52
CR0E - Text Cursor Location High Register....52
CR0F - Text Cursor Location Low Register....52
CR10 - Vertical Sync Start Register....53
CR11 - Vertical Sync End Register....54
CR12 - Vertical Display Enable End Register....55
CR13 - Offset Register....55
CR14 - Underline Location Register....56
CR15 - Vertical Blanking Start Register....57
CR16 - Vertical Blanking End Register....57
CR17 - CRT Mode Control....58

CR18 - Line Compare Register 62
CR22 - Memory Read Latch Data Register 63
CR24 - Toggle State of Attribute Controller Register 63
Display Audio Codec Verbs 64
Block Diagram 64
Codec Node Hierarchy 65
Programming 66
Verb Support 66
Parameter Support 68
Node ID 00h: Root Node Verbs 69
F00h - Get Parameters 69
Parameter 00h: VID - Vendor ID 69
Parameter 02h: RID - Revision ID 69
Parameter 04h: PARAM_SNC - Subordinate Node Count 69
Node ID 01h: Audio Function Group Verbs 70
F00h: Get Parameters 70
Parameter 04h: PARAM_SNC - Subordinate Node Count 70
Parameter 05h: PARAM_FGT - Function Group Type 70
Parameter 08h: PARAM_FGC - Function Group Capability 70
Parameter 0Fh: PARAM_SPS - Supported Power States 71
705h: SET_PS - Set Power State 71
F05h: GET_PS - Get Power State 71
F20h: GET SSID - Get Subsystem ID0 71
720h: SET SSID0 - Set Subsystem ID0 71
721h: SET SSID1 - Set Subsystem ID1 72
722h: SET SSID2 - Set Subsystem ID2 72
723h: SET SSID3 - Set Subsystem ID3 72
Node ID 02h, 03h, 04h: Audio Output Convertor Widget Verbs 73
2h/Ah: SET/GET_SDF - Set/GET Stream Descriptor Format 73
F00h: Get Parameters 74
Parameter 09h: AWC - Audio Widget Capabilities 74
Parameter 0Ah: PSB - PCM Sizes and Bit Rates 74
Parameter 0Bh: SF - Stream Formats 75

Parameter 0Fh: PARAM_SPS - Supported Power States....75
705h: SET_PS - Set Power State....75
F05h: GET_PS - Get Power State....76
706h/F06h: GET/SET_CSID - Get/Set Channel and Stream ID....76
Digital Converter Verbs....76
F0Dh: GET_DC - Get Digital Converter....76
70Dh: SET_DC1 - Set Digital Converter 1....77
70Eh: Digital Converter 2....77
73Eh: Digital Converter 3....77
73Fh: Digital Converter 4....77
72Dh/F2Dh: GET/SET_CCC - Get/Set Converter Channel Count....77
Node ID 05h, 06h, 07h: Pin Widget Verbs....78
3h: SET_AM - Set Amplifier Mute....78
B8h: GET_AM - Get Amplifier Mute....79
F00h: Get Parameters....79
Parameter 09h: AWC - Audio Widget Capabilities....79
Parameter 0Ch: PC - Pin Capabilities....80
Parameter 0Eh: CLL - Connection List Length....80
Parameter 12h: OAC - Output Amplifier Capabilities....80
Parameter 15h: DLL - Device List Length....80
Parameter 0Fh: PARAM_SPS - Supported Power States....81
701h/F01h: SET/GET_CSC - Set/Get Connection Select Control....81
F02h: GET_CLE - Get Connection List Entry....81
705h: SET_PS - Set Power State....81
F05h: GET_PS - Get Power State....81
707h/F07h: SET/GET_PWC - Set/Get Pin Widget Control....82
708h/F08h: SET/GET_UE - Set/Get Unsolicited Enable....82
F09h: GET_PS - Get Pin Sense....82
71Ch: SET_CD0 - Set Configuration Default Byte 0....82
71Dh: SET_CD1 - Set Configuration Default Byte 1....82
71Eh: SET_CD2 - Set Configuration Default Byte 2....82
71Fh: SET_CD3 - Set Configuration Default Byte 3....83
F1Ch: GET_CD - Get Configuration Default....83

F2Eh: HDMI/DP Info Size 83
F2Fh: Get ELD Data 84
Parameter nn: ELD Data 84
730h/F30h: SET/GET_HII - Set/Get HDMI Info Index 84
731h/F31h: SET/GET_HID - Set/Get HDMI Info Data 84
732h/F32h: SET/GET_HITC - Set/Get HDMI Info Transmit Control 84
733h SET_PC - Set Protection Control 85
734h/F34h: SET/GET_CCM - Get/Set Converter Channel Map 85
735h: SET_DS - Set Device Select 85
F35h: GET_DS - Get Device Select 85
F36h: GET_DDLE - Get Display Device List Entry 86
73Ch/F3Ch: SET/GET_DPID - Set/Get DisplayPort Stream ID 86
Node ID 08h: Intel Vendor Widget Verbs 87
F00h: Get Parameters 87
Parameter 09h: AWC - Audio Widget Capabilities 87
781h/F81h: GET/SET_VV - Get/Set HDMI Vendor Verb 87
F83h: GET_CGTC - Get Captured GTC Value 88
F84h: GET_CWC - Get Captured Wall Clock Value 88
F85h: GET GOF - Get GTC Offset Value 88
785h: SET GOF0 - Set GTC Offset Value Byte 0 88
786h: SET GOF1 - Set GTC Offset Value Byte 1 88
787h: SET GOF2 - Set GTC Offset Value Byte 2 88
788h: SET GOF3 - GTC Offset Value Byte 3 88
789h/F89h: SET/GET_GDI - Set/Get GTC Device Index 88

VGA and Extended VGA Registers

This section describes the registers and the functional operation notations for the observable registers in the VGA section. This functionality is provided as a means for support of legacy applications and operating systems. It is important to note that these registers in general have the desired effects only when running VGA display modes.

The main exceptions to this are the palette interface which allows real mode DOS applications and full screen VGA applications under an OS control running in high resolution (non-VGA) modes to access the palette through the VGA register mechanisms and the use of the ST01 status bits that determine when the VGA enters display enable and sync periods. Other exceptions include the register bits that control the memory accesses through the A000:0000 and B000:0000 memory segments which are used during operating system emulation of VGA for "DOS box" applications. Some of the functions of the VGA are enabled or defeated through the programming of the VGA control register bits that are located in the MMIO register space.

Given the legacy nature of this function, it has been adapted to the changing environment that it must operate within. The three most notable changes are the addition of high resolution display mode support, new operating system support, and the use of fixed resolution display devices (such as LCD panels). Additional control bits in the PCI Config space will affect the ability to access the registers and memory aperture associated with VGA.

Mode of Operation	VGA Disable	VGA Display	VGA Registers	Palette (VGA)	VGA Memory	VGA Banking
VGA DOS	No	Yes	Yes	Yes	Yes	No
HiRes DOS	Yes	No	Yes	Yes	No	Yes
Fullscreen DOS	Yes/No	No/Yes	Yes	Yes	Yes	Yes
DOS Emulation	Yes	No	Yes	Yes	Yes	Yes

VGA Display Mode	Dot Clock Select	Dot Clock Range	132 Column Text Support	9-Dot Disable Support	Main Use
Native	VGA Clock Select	25/28 MHz	No	No	Analog CRT (VGA connector)
Centered	Fixed at display Requirements	Product Specific	No	Yes	Digital Display
Upper Left Corner	Fixed at display Requirements	Product Specific	No	Yes	Internal Panel

Native, Centered, and Upper Left Corner support varies from product to product.

Even in the native VGA display operational modes, not all combinations of bit settings result in functional operating modes. VGA display modes have the restriction that they can be used only when all other display planes are disabled.

These registers are accessed via I/O space. The I/O space resides in the PCI compatibility hole and uses only the addresses that were part of the original VGA I/O space (which includes EGA and MDA emulation). Accesses to the VGA I/O addresses are steered to the proper bus and rely on proper setup of bridge registers. Extended VGA registers such as GR10 and GR11 use additional indexes for the already defined I/O addresses. VGA register accesses are allowed as 8 or 16 bit naturally aligned transactions only. Word transactions must have the least significant bit of the address set to zero. DWORD I/O operations should not be performed on these registers.

Some products may support access to these registers through MMIO. The access method varies and is documented elsewhere.

General Control and Status Registers

The setup, enable, and general registers are all directly accessible by the CPU. A sub indexing scheme is not used to read from and write to these registers.

Name	Function	Read		Write	
		I/O	Memory Offset	I/O	Memory Offset
ST00	VGA Input Status Register 0	3C2h	3C2h	--	--
ST01	VGA Input Status Register 1	3BAh/3DAh[1]	3BAh/3DAh[1]	--	--
FCR	VGA Feature Control Register	3CAh	3CAh	3BAh/3DAh[1]	3BAh/3DAh[1]
MSR	VGA Miscellaneous Output Register	3CCh	3CCh	3C2h	3C2h

[1] The address selection for ST01 reads and FCR writes is dependent on CGA or MDA emulation mode as selected via the MSR register.

Various bits in these registers provide control over the real-time status of the horizontal sync signal, the horizontal retrace interval, the vertical sync signal, and the vertical retrace interval. The horizontal retrace interval is the period during the drawing of each scan line containing active video data, when the active video data is not being displayed. This period includes the horizontal front and back porches, and the horizontal sync pulse. The horizontal retrace interval is always longer than the horizontal sync pulse. The vertical retrace interval is the period during which the scan lines not containing active video data are drawn. This includes the vertical front porch, back porch, and the vertical sync pulse. The vertical retrace interval is normally longer than the vertical sync pulse.

ST00 - Input Status 0

I/O (and Memory Offset) Address: 3C2h

Default: 00h

Attributes: Read Only

Bit	Descriptions
7	**CRT Interrupt Pending.** This bit is here for EGA compatibility and **will always return zero**. The generation of interrupts was originally enabled, through bits [4,5] of the Vertical Retrace End Register (CR11). This ability to generate interrupts at the start of the vertical retrace interval is a feature that is typically unused by DOS software and therefore is only supported through other means for use under a operating system support. 0 = CRT (vertical retrace interval) interrupt is not pending. 1 = CRT (vertical retrace interval) interrupt is pending
6:5	**Reserved.** Read as 0s.
4	**RGB Comparator / Sense.** This bit is here for compatibility and **will always return one**. Monitor detection must be done be done through the programming of registers in the MMIO space. 0 = Below threshold 1 = Above threshold
3:0	**Reserved.** Read as 0s.

ST01 - Input Status 1

I/O (and Memory Offset) Address: 3BAh/3DAh

Default: 00h

Attributes: Read Only

The address selection is dependent on CGA or MDA emulation mode as selected via the MSR register.

Bit	Descriptions
7	**Reserved (as per VGA specification).** Read as 0s.
6	**Reserved.** Read as 0.
5:4	**Video Feedback 1, 0.** These bits are connected to 2 of the 8 color bits sent to the palette. Bits 4 and 5 of the Color Plane Enable Register (AR12) selects which two of the 8 possible color bits become connected to these 2 bits of this register. These bits exist for EGA compatibility.
3	**Vertical Retrace/Video.** 0 = VSYNC inactive (Indicates that a vertical retrace interval is not taking place). 1 = VSYNC active (Indicates that a vertical retrace interval is taking place). VGA pixel generation is not locked to the display output but is loosely coupled. A VSYNC indication may not occur during the actual VSYNC going to the display but during the VSYNC that is generated as part of the VGA pixel generation. The exact relationship will vary with the VGA display operational mode. This status bit will remain active when the VGA is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now incorrectly) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. Bits 4 and 5 of the Vertical Retrace End Register (CR11) previously could program this bit to generate an interrupt at the start of the vertical retrace interval. This ability to generate interrupts at the start of the vertical retrace interval is a feature that is largely unused by legacy software. Interrupts are not supported through the VGA register bits.
2:1	**Reserved.** Read as 0s.
0	**Display Enable Output.** Display Enable is a status bit (bit 0) in VGA Input Status Register 1 that indicates when either a horizontal retrace interval or a vertical retrace interval is taking place. This bit was used with the EGA graphics system (and the ones that preceded it, including MDA and CGA). In those cases, it was important to check the status of this bit to ensure that one or the other retrace intervals was taking place before reading from or writing to the frame buffer. In these earlier systems, reading from or writing to the frame buffer at times outside the retrace intervals meant that the CRT controller would be denied access to the frame buffer. Those behaviors resulted in either "snow" or a flickering display. This bit provides compatibility with software

Bit	Descriptions
	designed for those early graphics controllers. This bit is currently used in DOS applications that access the palette to prevent the sparkle associated with read and write accesses to the palette RAM with the same address on the same clock cycle. **This status bit remains active when the VGA display is disabled and the device is running in high resolution modes (non-VGA) to allow for applications that (now considered incorrect) use these status registers bits. In this case, the status will come from the pipe that the VGA is assigned to. When in panel fitting VGA or centered VGA operation, the meaning of these bits will not be consistent with native VGA timings.** 0 = Active display data is being sent to the display. Neither a horizontal retrace interval or a vertical retrace interval is currently taking place. 1 = Either a horizontal retrace interval (horizontal blanking) or a vertical retrace interval (vertical blanking) is currently taking place.

FCR - Feature Control

I/O (and Memory Offset) Address: 3BAh/3DAh - Write; 3CAh - Read

Default: 00h

Attributes: See Address above

The I/O address used for writes is dependent on CGA or MDA emulation mode as selected via the MSR register. In the original EGA, bits 0 and 1 were used as part of the feature connector interface. Feature connector is not supported in these devices and those bits will always read as zero.

Bit	Descriptions
7:4	**Reserved.** Read as 0.
3	**VSYNC Control.** This bit is provided for compatibility only and has no other function. Reads and writes to this bit have no effect other than to change the value of this bit. The previous definition of this bit selected the output on the VSYNC pin. 0 = Was used to set VSYNC output on the VSYNC pin (default). 1 = Was used to set the logical 'OR' of VSYNC and Display Enable output on the VSYNC pin. This capability was not typically very useful.
2:0	**Reserved.** Read as 0.

MSR - Miscellaneous Output

I/O (and Memory Offset) Address: 3C2h - Write; 3CCh - Read

Default: 00h

Attributes: See Address above

Bit	Descriptions
7	**CRT VSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. Sync polarity was used in VGA to signal the monitor how many lines of active display are being generated. 0 = Positive Polarity (default). 1 = Negative Polarity.
6	**CRT HSYNC Polarity.** This is a legacy function that is used in native VGA modes. For most cases, sync polarity will be controlled by the port control bits. The VGA settings can be optionally selected for compatibility with the original VGA when used in the VGA native mode. 0 = Positive Polarity (default). 1 = Negative Polarity
5	**Page Select.** In Odd/Even Memory Map Mode 1 (GR6), this bit selects the upper or lower 64 KB page in display memory for CPU access: 0 = Upper page (default) 1 = Lower page. Selects between two 64KB pages of frame buffer memory during standard VGA odd/even modes (modes 0h through 5h). Bit 1 of register GR06 can also program this bit in other modes. This bit is would normally set to 1 by the software.
4	**Reserved.** Read as 0.
3:2	**Clock Select.** These bits can select the dot clock source for the CRT interface. The bits should be used to select the dot clock in standard native VGA modes only. When in the centering or upper left corner modes, these bits should be set to have no effect on the clock rate. The actual frequencies that these bits select, if they have any affect at all, is programmable through the DPLL MMIO registers. 00 = CLK0, 25.175 MHz (for standard VGA modes with 640 pixel (8-dot) horizontal resolution) (default) 01 = CLK1, 28.322 MHz. (for standard VGA modes with 720 pixel (9-dot) horizontal resolution) 10 = Was used to select an external clock (now unused) 11 = Reserved
1	**A0000-BFFFFh Memory Access Enable.** VGA Compatibility bit enables access to video memory (frame buffer) at A0000-BFFFFh. When disabled, accesses to VGA memory are blocked in this region. This bit is independent of and does not block CPU access to the video linear frame buffer at other addresses.

Bit	Descriptions
	0 = Prevent CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture (default). 1 = Allow CPU access to memory/registers/ROM through the A0000-BFFFF VGA memory aperture. Delete?
0	**I/O Address Select.** This bit selects 3Bxh or 3Dxh as the I/O address for the CRT Controller registers, the Feature Control Register (FCR), and Input Status Register 1 (ST01). Presently ignored (whole range is claimed), but will "ignore" 3Bx for color configuration or 3Dx for monochrome. It is typical in AGP chipsets to shadow this bit and properly steer I/O cycles to the proper bus for operation where a MDA exists on another bus such as ISA. 0 = Select 3Bxh I/O address (MDA emulation) (default). 1 = Select 3Dxh I/O address (CGA emulation).

In standard VGA modes using the analog VGA connector, bits 7 and 6 indicate which of the three standard VGA vertical resolutions the standard VGA display should use. Extended modes, including those with a vertical resolution of 480 scan lines, may use a setting of 0 for both of these bits. Different connector standards and timing standards specify the proper use of sync polarity. This setting was "reserved" in the VGA standard.

Analog CRT Display Sync Polarities

V	H	Display	Horizontal Frequency	Vertical Frequency
P	P	200 Line	15.7 KHz	60 Hz
N	P	350 Line	21.8 KHz	60 Hz
P	N	400 Line	31.5 KHz	70 Hz
N	N	480 Line	31.5 KHz	60 Hz

Sequencer Registers

The sequencer registers are accessed via either I/O space or Memory space. To access registers the VGA Sequencer Index register (SRX) at I/O address 3C4h (or memory address 3C4h) is written with the index of the desired register. Then the desired register is accessed through the data port for the sequencer registers at I/O address 3C5 (or memory address 3C5).

SRX - Sequencer Index

I/O (and Memory Offset) Address: 3C4h

Default: 00h

Attributes: Read/Write

Bit	Description
7:3	**Reserved.** Read as 0s.
2:0	**Sequencer Index.** This field contains a 3-bit Sequencer Index value used to access sequencer data registers at indices 0 through 7.

SR00 - Sequencer Reset

I/O (and Memory Offset) Address: 3C5h(Index=00h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:2	**Reserved.** Read as 0.
1	**Reserved.** Reserved for VGA compatibility (was reset).
0	**Reserved.** Reserved for VGA compatibility. (was reset)

SR01 - Clocking Mode

I/O (and Memory Offset) Address: 3C5h (Index=01h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:6	**Reserved.** Read as 0s.
5	**Screen Off.** 0 = Normal Operation (default). 1 = Disables video output (blanks the screen) and turns off display data fetches. Synchronization pulses to the display, however, are maintained. Setting this bit to 1 had been used as a way to more rapidly update and improve CPU access performance to the frame buffer during VGA modes. In non-VGA modes (VGA Disable=1), this bit has no effect. Before the VGA is disabled through the MMIO VGA control register, this bit should be set to stop the memory accesses from the display. The following sequence must be used when disabling the VGA plane. 1. Write SR01 to set bit 5 = 1 to disable video output. 2. Wait for 100us. 3. Disable the VGA plane via Bit 31 of the MMIO VGA control register (location found in the MMIO display register programming specification).
4	**Shift 4.** 0 = Load video shift registers every 1 or 2 character clocks (depending on bit 2 of this register) (default). 1 = Load shift registers every 4th character clock.
3	**Dot Clock Divide.** Setting this bit to 1 stretches doubles all horizontal timing periods that are specified in the VGA horizontal CRTC registers. This bit is used in standard VGA 40-column text modes to stretch timings to create horizontal resolutions of either 320 or 360 pixels (as opposed to 640 or 720 pixels, normally used in standard VGA 80-column text modes). The effect of this is that there will actually be twice the number of pixels sent to the display per line. 0 = Pixel clock is left unaltered (used for 640 (720) pixel modes); (default). 1 = Pixel clock divided by 2 (used for 320 (360) pixel modes).
2	**Shift Load.** Bit 4 of this register must be 0 for this bit to be effective. 0 = Load video data shift registers every character clock (default).

Bit	Descriptions
	1 = Load video data shift registers every other character clock.
1	**Reserved.** Read as 0.
0	**8/9 Dot Clocks.** This bit determines whether a character clock is 8 or 9 dot clocks long if clock doubling is disabled and 16 or 18 clocks if it is. This also changes the interpretation of the pixel panning values (see chart). An additional control bit determines if this bit is to be ignored and 8-dot characters are to be used always. The 9-dot disable would be used when doubling the horizontal pixels on a 1280 wide display or non-doubling on a 640 wide display. Panning however will occur according to the expected outcome. 0 = 9 dot clocks (9 horizontal pixels) per character in text modes with a horizontal resolution of 720 pixels. 1 = 8 dot clocks (8 horizontal pixels) per character in text or graphics modes with a horizontal resolution of 640 pixels.

SR02 - Plane/Map Mask

I/O (and Memory Offset) Address: 3C5h (Index=02h)

Default: 00h

Attributes: Read/Write

Bit	Descriptions
7:4	**Reserved.** Read as 0s.
3:0	**Memory Planes [3:0] Processor Write Access Enable.** In both the Odd/Even Mode and the Chain 4 Mode, these bits still control access to the corresponding color plane. 0 = Disable. 1 = Enable. This register is referred to in the VGA standard as the Map Mask Register.

SR03 - Character Font

I/O (and Memory Offset) Address: 3C5h (index=03h)

Default: 00h

Attributes: Read/Write

In text modes, bit 3 of the video data's attribute byte normally controls the foreground intensity. This bit may be redefined to control switching between character sets. This latter function is enabled whenever there is a difference in the values of the Character Font Select A and the Character Font Select B bits. If the two values are the same, the character select function is disabled and attribute bit 3 controls the foreground intensity.

Bit 1 of the Memory Mode Register (SR04) must be set to 1 for the character font select function of this register to be active. Otherwise, only character maps 0 and 4 are available.

<table>
<tr><th>Bit</th><th>Descriptions</th></tr>
<tr><td>7:6</td><td>Reserved. Read as 0s.</td></tr>
<tr><td>3:2,5</td><td>Character Map Select Bits for Character Map B. These three bits are used to select the character map (character generator tables) to be used as the secondary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [3:2, 5]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
<tr><td>10,0</td><td>2</td><td>5th 8KB of plane 2 at offset 32K</td></tr>
<tr><td>10,1</td><td>6</td><td>6th 8KB of plane 2 at offset 40K</td></tr>
<tr><td>11,0</td><td>3</td><td>7th 8KB of plane 2 at offset 48K</td></tr>
<tr><td>11,1</td><td>7</td><td>8th 8KB of plane 2 at offset 56K</td></tr>
</table></td></tr>
<tr><td>1:0,4</td><td>Character Map Select Bits for Character Map A. These three bits are used to select the character map (character generator tables) to be used as the primary character set (font). The numbering of the maps is not sequential.
<table>
<tr><th>Bit [1:0,4]</th><th>Map Number</th><th>Table Location</th></tr>
<tr><td>00,0</td><td>0</td><td>1st 8KB of plane 2 at offset 0 (default)</td></tr>
<tr><td>00,1</td><td>4</td><td>2nd 8KB of plane 2 at offset 8K</td></tr>
<tr><td>01,0</td><td>1</td><td>3rd 8KB of plane 2 at offset 16K</td></tr>
<tr><td>01,1</td><td>5</td><td>4th 8KB of plane 2 at offset 24K</td></tr>
</table></td></tr>
</table>

Bit	Descriptions		
	10,0	2	5th 8KB of plane 2 at offset 32K
	10,1	6	6th 8KB of plane 2 at offset 40K
	11,0	3	7th 8KB of plane 2 at offset 48K
	11,1	7	8th 8KB of plane 2 at offset 56K

SR04 - Memory Mode Register

I/O (and Memory Offset) Address: 3C5h (index=04h)

Default: 00h

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3	**Chain 4 Mode.** The selections made by this bit affect both CPU Read and write accesses to the frame buffer. 0 = The manner in which the frame buffer memory is mapped is determined by the setting of bit 2 of this register (default). 1 = The frame buffer memory is mapped in such a way that the function of address bits 0 and 1 are altered so that they select planes 0 through 3. This setting is used in mode x13 to allow all four planes to be accessed via sequential addresses.
2	**Odd/Even Mode.** Bit 3 of this register must be set to 0 for this bit to be effective. The selections made by this bit affect only non-paged CPU accesses to the frame buffer through the VGA aperture. 0 = The frame buffer memory is mapped in such a way that the function of address bit 0 such that even addresses select planes 0 and 2 and odd addresses select planes 1 and 3 (default). 1 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02).
1	**Extended Memory Enable.** This bit must be set to 1 to enable the selection and use of character maps in plane 2 via the Character Map Select Register (SR03). 0 = Disable CPU accesses to more than the first 64KB of VGA standard memory (default). 1 = Enable CPU accesses to the rest of the 256KB total VGA memory beyond the first 64KB.
0	**Reserved.** Read as 0.

SR07 - Horizontal Character Counter Reset

I/O (and Memory Offset) Address: 3C5h (index=07h)

Default: 00h

Attributes: Read/Write

For standard VGAs, writing this register (with any data) causes the horizontal character counter to be held in reset (the character counter output will remain 0). It remained in reset until a write occurred to any other sequencer register location with SRX set to an index of 0 through 6. In this implementation that sequence has no such special effect.

The vertical line counter is clocked by a signal derived from the horizontal display enable (which does not occur if the horizontal counter is held in reset). Therefore, if a write occurs to this register during the vertical retrace interval, both the horizontal and vertical counters will be set to 0. A write to any other sequencer register location (with SRX set to an index of 0 through 6) may then be used to start both counters with reasonable synchronization to an external event via software control. Although this was a standard VGA register, it was not documented.

Bit	Description
7:0	**Horizontal Character Counter.**

Graphics Controller Registers

The graphics controller registers are accessed via either I/O space or Memory space. Accesses to the registers of the VGA Graphics Controller are done through the use of address 3CEh (or memory address 3CEh) written with the index of the desired register. Then the desired register is accessed through the data port for the graphics controller registers at I/O address 3CFh (or memory address 3CFh). Indexes 10 and 11 should only be accessed through the I/O space only.

GRX - GRX Graphics Controller Index Register

I/O (and Memory Offset) Address: 3CEh

Default: 000UUUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:0	**Graphics Controller Register Index.** This field selects any one of the graphics controller registers (GR00-GR18) to be accessed via the data port at I/O (or memory offset) location 3CFh.

GR00 - Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (index=00h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Set/Reset Plane [3:0].** When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 0, all 8 bits of each byte of each memory plane are set to either 1 or 0 as specified in the corresponding bit in this register, if the corresponding bit in the Enable Set/Reset Register (GR01) is set to 1. When the Write Mode bits (bits 0 and 1) of the Graphics Mode Register (GR05) are set to select Write Mode 3, all CPU data written to the frame buffer is rotated, then logically ANDed with the contents of the Bit Mask Register (GR08), and then treated as the addressed data's bit mask, while value of these four bits of this register are treated as the color value.

GR01 - Enable Set/Reset Register

I/O (and Memory Offset) Address: 3CFh (Index=01h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Enable Set/Reset Plane [3:0].** This register works in conjunction with the Set/Reset Register (GR00). The Write Mode bits (bits 0 and 1) must be set for Write Mode 0 for this register to have any effect. 0 = The corresponding memory plane can be read from or written to by the CPU without any special bitwise operations taking place. 1 = The corresponding memory plane is set to 0 or 1 as specified in the Set/Reset Register (GR00).

GR02 - Color Compare Register

I/O (and Memory Offset) Address: 3CFh (Index=02h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Color Compare Plane [3:0].** When the Read Mode bit (bit 3) of the Graphics Mode Register (GR05) is set to select Read Mode 1, all 8 bits of each byte of each of the 4 memory planes of the frame buffer corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison, wherein value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.

GR03 - Data Rotate Register

I/O (and Memory Offset) Address: 3CFh (Index=03h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:5	**Reserved.** Read as 0.
4:3	**Function Select.** These bits specify the logical function (if any) to be performed on data that is meant to be written to the frame buffer (using the contents of the memory read latch) just before it is actually stored in the frame buffer at the intended address location. 00 = Data being written to the frame buffer remains unchanged, and is simply stored in the frame buffer. 01 = Data being written to the frame buffer is logically ANDed with the data in the memory read latch before it is actually stored in the frame buffer. 10 = Data being written to the frame buffer is logically ORed with the data in the memory read latch before it is actually stored in the frame buffer. 11 = Data being written to the frame buffer is logically XORed with the data in the memory read latch before it is actually stored in the frame buffer.
2:0	**Rotate Count.** These bits specify the number of bits to the right to rotate any data that is meant to be written to the frame buffer just before it is actually stored in the frame buffer at the intended address location.

GR04 - Read Plane Select Register

I/O (and Memory Offset) Address: 3CFh (Index=04h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**Read Plane Select.** These two bits select the memory plane from which the CPU reads data in Read Mode 0. In Odd/Even Mode, bit 0 of this register is ignored. In Chain 4 Mode, both bits 1 and 0 of this register are ignored. The four memory planes are selected as follows: 00 = Plane 0 01 = Plane 1 10 = Plane 2 11 = Plane 3 These two bits also select which of the four memory read latches may be read via the Memory read Latch Data Register (CR22). The choice of memory read latch corresponds to the choice of plane specified in the table above. The Memory Read Latch Data register and this additional function served by 2 bits are features of the VGA standard that were never documented.

GR05 - Graphics Mode Register

I/O (and Memory Offset) Address: 3CFh (Index=05h)

Default: 0UUU U0UUb (U=Undefined)

Attributes: Read/Write

<table>
<tr><th>Bit</th><th>Description</th></tr>
<tr><td>7</td><td>Reserved. Read as 0.</td></tr>
<tr><td>6:5</td><td>
Shift Register Control. In standard VGA modes, pixel data is transferred from the 4 graphics memory planes to the palette via a set of 4 serial output bits. These 2 bits of this register control the format in which data in the 4 memory planes is serialized for these transfers to the palette.

Bits [6:5]=00

One bit of data at a time from parallel bytes in each of the 4 memory planes is transferred to the palette via the 4 serial output bits, with 1 of each of the serial output bits corresponding to a memory plane. This provides a 4-bit value on each transfer for 1 pixel, making possible a choice of 1 of 16 colors per pixel.

Serial
<table>
<tr><th>Out</th><th>1st Xfer</th><th>2nd Xfer</th><th>3rd Xfer</th><th>4th Xfer</th><th>5th Xfer</th><th>6th Xfer</th><th>7th Xfer</th><th>8th Xfer</th></tr>
<tr><td>Bit 3</td><td>plane3 bit7</td><td>plane3 bit6</td><td>plane3 bit5</td><td>plane3 bit4</td><td>plane3 bit3</td><td>plane3 bit2</td><td>plane3 bit1</td><td>plane3 bit0</td></tr>
<tr><td>Bit 2</td><td>plane2 bit7</td><td>plane2 bit6</td><td>plane2 bit5</td><td>plane2 bit4</td><td>plane2 bit3</td><td>plane2 bit2</td><td>plane2 bit1</td><td>plane2 bit0</td></tr>
<tr><td>Bit 1</td><td>plane1 bit7</td><td>plane1 bit6</td><td>plane1 bit5</td><td>plane1 bit4</td><td>plane1 bit3</td><td>plane1 bit2</td><td>plane1 bit1</td><td>plane1 bit0</td></tr>
<tr><td>Bit 0</td><td>plane0 bit7</td><td>plane0 bit6</td><td>plane0 bit5</td><td>plane0 bit4</td><td>plane0 bit3</td><td>plane0 bit2</td><td>plane0 bit1</td><td>plane0 bit0</td></tr>
</table>

Bits [6:5]=01

Two bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that alternates per byte between memory planes 0 and 2, and memory planes 1 and 3. First the even-numbered and odd-numbered bits of a byte in memory plane 0 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of a byte in memory plane 2 are transferred via serial output bits 2 and 3. Next, the even-numbered and odd-numbered bits of a byte in memory plane 1 are transferred via serial output bits 0 and 1, respectively, while the even-numbered and odd-numbered bits of memory plane 3 are transferred via serial out bits 1 and 3. This provides a pair of 2-bit values (one 2-bit value for each of 2 pixels) on each transfer,
</td></tr>
</table>

Bit	Description
	making possible a choice of 1 of 4 colors per pixel. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane2 bit7 plane2 bit5 plane2 bit3 plane2 bit1 plane3 bi7t plane3 bit5 plane3 bit3 plane3 bit1 Bit 2 plane2 bit6 plane2 bit4 plane2 bit2 plane2 bit0 plane3 bit6 plane3 bit4 plane3 bit2 plane3 bit0 Bit 1 plane0 bit7 plane0 bit5 plane0 bit3 plane0 bit1 plane1 bit7 plane1 bit5 plane1 bit3 plane1 bit1 Bit 0 plane0 bit6 plane0 bit4 plane0 bit2 plane0 bit0 plane1 bit6 plane1 bit4 plane1 bit2 plane1 bit0 This alternating pattern is meant to accommodate the use of the Odd/Even mode of organizing the 4 memory planes, which is used by standard VGA modes 2h and 3h. **Bits [6:5]=1x** Four bits of data at a time from parallel bytes in each of the 4 memory planes are transferred to the palette in a pattern that iterates per byte through memory planes 0 through 3. First the 4 most significant bits of a byte in memory plane 0 are transferred via the 4 serial output bits, followed by the 4 least significant bits of the same byte. Next, the same transfers occur from the parallel byte in memory planes 1, 2 and lastly, 3. Each transfer provides either the upper or lower half of an 8 bit value for the color for each pixel, making possible a choice of 1 of 256 colors per pixel. This is the setting used in mode x13. Serial Out 1st Xfer 2nd Xfer 3rd Xfer 4th Xfer 5th Xfer 6th Xfer 7th Xfer 8th Xfer Bit 3 plane0 bit7 plane0 bit3 plane1 bit7 plane1 bit3 plane2 bit7 plane2 bit3 plane3 bit7 plane3 b3it Bit 2 plane0 bit6 plane0 bit2 plane1 bit6 plane1 bit2 plane2 bit6 plane2 bit2 plane3 bit6 plane3 bit2 Bit 1 plane0 bit5 plane0 bit1 plane1 bit5 plane1 bit1 plane2 bit5 plane2 bit1 plane3 bit5 plane3 bit1 Bit 0 plane0 bit4 plane0 bit0 plane1 bit4 plane1 bit0 plane2 bit4 plane2 bit0 plane3 bit4 plane3 bit0 This pattern is meant to accommodate mode 13h, a standard VGA 256-color

Bit	Description
	graphics mode.
4	**Odd/Even Mode.** 0 = Addresses sequentially access data within a bit map, and the choice of which map is accessed is made according to the value of the Plane Mask Register (SR02). 1 = The frame buffer is mapped in such a way that the function of address bit 0 is such that even addresses select memory planes 0 and 2 and odd addresses select memory planes 1 and 3. This works in a way that is the inverse of (and is normally set to be the opposite of) bit 2 of the Memory Mode Register (SR02).
3	**Read Mode.** 0 = During a CPU read from the frame buffer, the value returned to the CPU is data from the memory plane selected by bits 1 and 0 of the Read Plane Select Register (GR04). 1 = During a CPU read from the frame buffer, all 8 bits of the byte in each of the 4 memory planes corresponding to the address from which a CPU read access is being performed are compared to the corresponding bits in this register (if the corresponding bit in the Color Don't Care Register (GR07) is set to 1). The value that the CPU receives from the read access is an 8-bit value that shows the result of this comparison. A value of 1 in a given bit position indicates that all of the corresponding bits in the bytes across all 4 of the memory planes that were included in the comparison had the same value as their memory plane's respective bits in this register.
2	**Reserved.** Read as 0.
1:0	**Write Mode.** 00 = Write Mode 0 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written with the CPU write data after it has been rotated by the number of counts specified in the Data Rotate Register (GR03). If, however, the bit(s) in the Enable Set/Reset Register (GR01) corresponding to one or more of the memory planes is set to 1, then those memory planes will be written to with the data stored in the corresponding bits in

Bit	Description
	the Set/Reset Register (GR00). 01 = Write Mode 1 - During a CPU write to the frame buffer, the addressed byte in each of the 4 memory planes is written to with the data stored in the memory read latches. (The memory read latches stores an unaltered copy of the data last read from any location in the frame buffer.) 10 = Write Mode 2 - During a CPU write to the frame buffer, the least significant 4 data bits of the CPU write data is treated as the color value for the pixels in the addressed byte in all 4 memory planes. The 8 bits of the Bit Mask Register (GR08) are used to selectively enable or disable the ability to write to the corresponding bit in each of the 4 memory planes that correspond to a given pixel. A setting of 0 in a bit in the Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with value of their counterparts in the memory read latches. A setting of 1 in a Bit Mask Register at a given bit position causes the bits in the corresponding bit positions in the addressed byte in all 4 memory planes to be written with the 4 bits taken from the CPU write data to thereby cause the pixel corresponding to these bits to be set to the color value. 11 = Write Mode 3 - During a CPU write to the frame buffer, the CPU write data is logically ANDed with the contents of the Bit Mask Register (GR08). The result of this ANDing is treated as the bit mask used in writing the contents of the Set/Reset Register (GR00) are written to addressed byte in all 4 memory planes.

GR06 - Miscellaneous Register

I/O (and Memory Offset) Address: 3CFh (Index=06h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0s.
3:2	**Memory Map Mode.** These 2 bits control the mapping of the VGA address range for frame buffer into the CPU address space as follows: 00 = A0000h - BFFFFh 01 = A0000h - AFFFFh 10 = B0000h - B7FFFh 11 = B8000h - BFFFFh This function is used in standard VGA modes, extended VGA modes (132 column text), and in non-VGA modes (hi-res). 132 column text modes are no longer supported. VGA aperture memory accesses are also controlled by the PCI configuration Memory Enable bit and MSR<1>. For accesses using GR10 and GR11 to paged VGA RAM or to device MMIO registers, set these bits to 01 to select the (A0000-AFFFF) range. The CPU must map this memory as uncacheable (UC).
1	**Chain Odd/Even.** This bit provides the ability to alter the interpretation of address bit A0, so that it may be used in selecting between the odd-numbered memory planes (planes 1 and 3) and the even-numbered memory planes (planes 0 and 2). 0 = A0 functions normally. 1 = A0 is switched with a high order address bit, in terms of how it is used in address decoding. The result is that A0 is used to determine which memory plane is being accessed (A0=0 for planes 0 and 2 and A0=1 for planes 1 and 3).
0	**Graphics/Text Mode.** This is one of two bits that are used to determine if the VGA is operating in text or graphics modes. The other bit is in AR10[0], these two bits need to be programmed in a consistent manner to achieve the proper results. 0 = Text mode. 1 = Graphics mode.

GR07 - Color Don't Care Register

I/O (and Memory Offset) Address: 3CFh (Index=07h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.** Read as 0.
3:0	**Ignore Color Plane [3:0].** These bits have effect only when bit 3 of the Graphics Mode Register (GR05) is set to 1 to select read mode 1. 0 = The corresponding bit in the Color Compare Register (GR02) will not be included in color comparisons. 1 = The corresponding bit in the Color Compare Register (GR02) is used in color comparisons.

GR08 - Bit Mask Register

I/O (and Memory Offset) Address: 3CFh (Index=08h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Bit Mask.** 0 = The corresponding bit in each of the 4 memory planes is written to with the corresponding bit in the memory read latches. 1 = Manipulation of the corresponding bit in each of the 4 memory planes via other mechanisms is enabled. This bit mask applies to any writes to the addressed byte of any or all of the 4 memory planes, simultaneously. This bit mask is applicable to any data written into the frame buffer by the CPU, including data that is also subject to rotation, logical functions (AND, OR, XOR), and Set/Reset. To perform a proper read-modify-write cycle into frame buffer, each byte must first be read from the frame buffer by the CPU (and this will cause it to be stored in the memory read latches), this Bit Mask Register must be set, and the new data then written into the frame buffer by the CPU.

Attribute Controller Registers

Unlike the other sets of indexed registers, the attribute controller registers are not accessed through a scheme employing entirely separate index and data ports. I/O address 3C0h (or memory address 3C0h) is used both as the read and write for the index register, and as the write address for the data port. I/O address 3C1h (or memory address 3C1h) is the read address for the data port.

To write to the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is written to the very same I/O (memory) address. A flip-flop alternates with each write to I/O address 3C0h (or memory address 3C0h) to change its function from writing the index to writing the actual data, and back again. This flip-flop may be deliberately set so that I/O address 3C0h (or memory address 3C0h) is set to write to the index (which provides a way to set it to a known state) by performing a read operation from Input Status Register 1 (ST01) at I/O address 3BAh (or memory address 3BAh) or 3DAh (or memory address 3DAh), depending on whether the graphics system has been set to emulate an MDA or a CGA as per MSR[0].

To read from the attribute controller registers, the index of the desired register must be written to I/O address 3C0h (or memory address 3C0h), and then the data is read from I/O address 3C1h (or memory address 3C1h). A read operation from I/O address 3C1h (or memory address 3C1h) does not reset the flip-flop to writing to the index. Only a write to 3C0h (or memory address 3C0h) or a read from 3BAh or 3DAh (or memory address 3BAh or 3DAh), as described above, will toggle the flip-flop back to writing to the index.

ARX - Attribute Controller Index Register

I/O (and Memory Offset) Address: 3C0h

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0s.
5	**Video Enable.** In the VGA standard, this is called the "Palette Address Source" bit. Clearing this bit will cause the VGA display data to become all 00 index values. For the default palette, this will cause a black screen. The video timing signals continue. Another control bit will turn video off and stop the data fetches. 0 = Disable. Attribute controller color registers (AR[00:0F]) can be accessed by the CPU. 1 = Enable. Attribute controller color registers (AR[00:0F]) are inaccessible by the CPU.
4:0	**Attribute Controller Register Index.** These five bits are used to select any one of the attribute controller registers (AR[00:14]), to be accessed.

AR[00:0F] - Palette Registers [0:F]

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=00h-0Fh)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:0	**Palette Bits P[5:0].** In each of these 16 registers, these are the lower 6 of 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors available to be selected in the palette. Bits 3 and 2 of the Color Select Register (AR14) supply bits P7 and P6 for the values contained in all 16 of these registers. Bits 1 and 0 of the Color Select Register (AR14) can also replace bits P5 and P4 for the values contained in all 16 of these registers, if bit 7 of the Mode Control Register (AR10) is set to 1.

AR10 - Mode Control Register

I/O (and Memory Offset) Address:	Read at 3C1h and Write at 3C0h; (index=10h)
Default:	UUh (U=Undefined)
Attributes:	Read/Write

Bit	Description
7	**Palette Bits P5, P4 Select.** **0** = P5 and P4 for each of the 16 selected colors (for modes that use 16 colors) are individually provided by bits 5 and 4 of their corresponding Palette Registers (AR[00:0F]). **1** = P5 and P4 for all 16 of the selected colors (for modes that use 16 colors) are provided by bits 1 and 0 of Color Select Register (AR14).
6	**Pixel Width/Clock Select.** **0** = Six bits of video data (translated from 4 bits via the palette) are output every dot clock. **1** = Two sets of 4 bits of data are assembled to generate 8 bits of video data which is output every other dot clock, and the Palette Registers (AR[00:0F]) are bypassed. This bit is set to 0 for all of the standard VGA modes, except mode 13h.
5	**Pixel Panning Compatibility.** **0** = Scroll both the upper and lower screen regions horizontally as specified in the Pixel Panning Register (AR13). **1** = Scroll only the upper screen region horizontally as specified in the Pixel Panning Register (AR13). This bit has application only when split-screen mode is being used, where the display area is divided into distinct upper and lower regions which function somewhat like separate displays.
4	**Reserved.** Read as 0.
3	**Enable Blinking/Select Background Intensity.** **0** = Disables blinking in graphics modes, and for text modes, sets bit 7 of the character attribute bytes to control background intensity, instead of blinking. **1** = Enables blinking in graphics modes and for text modes, sets bit 7 of the character attribute bytes to control blinking, instead of background intensity. The blinking rate is derived by dividing the VSYNC signal. The Blink Rate Control field of the VGA control register defines the blinking rate.
2	**Enable Line Graphics Character Code.** **0** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the background of the character of which

Bit	Description
	the given pixel is a part. **1** = Every 9th pixel of a horizontal line (i.e., the last pixel of each horizontal line of each 9-pixel wide character box) is assigned the same attributes as the 8th pixel if the character of which the given pixel is a part. This setting is intended to accommodate the line-drawing characters of the PC's extended ASCII character set -- characters with an extended ASCII code in the range of B0h to DFh. In some literature describing the VGA standard, the range of extended ASCII codes that are said to include the line-drawing characters is mistakenly specified as C0h to DFh, rather than the correct range of B0h to DFh.
1	**Select Display Type.** **0** = Attribute bytes in text modes are interpreted as they would be for a color display. **1** = Attribute bytes in text modes are interpreted as they would be for a monochrome display.
0	**Graphics/Alphanumeric Mode.** This bit (along with GR06[0]) select either graphics mode or text mode. These two bits must be programmed in a consistent manner to achieve the desired results. **0** = Alphanumeric (text) mode. **1** = Graphics mode.

AR11 - Overscan Color Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=11h)

Default: UUh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:0	**Overscan.** These 8 bits select the overscan (border) color index value. The actual border color will be determined by the contents of the palette at the selected index. The border color is displayed between the end of active and the beginning of blank or the end of blank and the beginning of active on CRT type devices driven from the DAC output port. For native VGA modes on digital display ports there is the option of including the border in the active region or not depending on a control bit in the port control register. For centered VGA modes, the VGA control register determines if the border is included in the centered region or not. For monochrome displays, this value should be set to 00h.

AR12 - Memory Plane Enable Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=12h)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5:4	**Video Status Mux.** These 2 bits are used to select 2 of the 8 possible palette bits (P7-P0) to be made available to be read via bits 5 and 4 of the Input Status Register 1 (ST01). The table below shows the possible choices. **Bit [5:4]** / **ST01 Bit 5** / **ST01 Bit 4** 00 / P2 (default) / P0 (default) 01 / P5 / P4 10 / P3 / P1 11 / P7 / P6 These bits are typically unused by current software; they are provided for EGA compatibility.
3:0	**Enable Plane [3:0].** These 4 bits individually enable the use of each of the 4 memory planes in providing 1 of the 4 bits used in video output to select 1 of 16 possible colors from the palette to be displayed. 0 = Disable the use of the corresponding memory plane in video output to select colors, forcing the bit that the corresponding memory plane would have provided to a value of 0. 1 = Enable the use of the corresponding memory plane in video output to select colors. AR12 is referred to in the VGA standard as the Color Plane Enable Register.

AR13 - Horizontal Pixel Panning Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=13h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:0	**Horizontal Pixel Shift 3-0.** This field holds a 4-bit value that selects the number of pixels by which the image is shifted horizontally to the left. This function is available in both text and graphics modes and allows for pixel panning. In text modes with a 9-pixel wide character box, the image can be shifted up to 9 pixels to the left. In text modes with an 8-pixel wide character box, and in graphics modes other than those with 256 colors, the image can be shifted up to 8 pixels to the left. A pseudo 9-bit mode is when the 9-dot character is selected but overridden by the VGA control bit. In standard VGA mode 13h (where bit 6 of the Mode Control Register, AR10, is set to 1 to support 256 colors), bit 0 of this register must remain set to 0, and the image may be shifted up to only 4 pixels to the left. In this mode, the number of pixels by which the image is shifted can be further controlled using bits 6 and 5 of the Preset Row Scan Register (CR08).

Number of Pixels Shifted				
Bits [3:0]	**9-dot**	**Pseudo 9-dot**	**8-dot**	**256-Color**
0	1	1	0	0
1	2	2	1	Undefined
2	3	3	2	1
3	4	4	3	Undefined
4	5	5	4	2
5	6	6	5	Undefined
6	7	7	6	3
7	8	7	7	Undefined
8	0	0	Undefined	Undefined

AR14 - Color Select Register

I/O (and Memory Offset) Address: Read at 3C1h and Write at 3C0h; (index=14h)

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7:4	**Reserved.**
3:2	**Palette Bits P[7:6].** These are the 2 upper-most of the 8 bits that are used to map either text attributes or pixel color input values (for modes that use 16 colors) to the 256 possible colors contained in the palette. These 2 bits are common to all 16 sets of bits P5 through P0 that are individually supplied by Palette Registers 0-F (AR[00:0F]).
1:0	**Alternate Palette Bits P[5:4].** These 2 bits can be used as an alternate version of palette bits P5 and P4. Unlike the P5 and P4 bits that are individually supplied by Palette Registers 0-F (AR[00:0F]), these 2 alternate palette bits are common to all 16 of Palette Registers. Bit 7 of the Mode Control Register (AR10) is used to select between the use of either the P5 and P4 bits that are individually supplied by the 16 Palette Registers or these 2 alternate palette bits.

VGA Color Palette Registers

In devices that have multiple display pipes, there is one palette for each display pipe. These palettes are the same for VGA modes and non-VGA modes. Accesses through VGA register methods can optionally read or write from either one.

For each palette, the color data stored in these 256 color data positions can be accessed only through a complex sub-addressing scheme, using a data register and two index registers. The Palette Data Register at I/O address 3C9h (or memory address offset 3C1h) is the data port. The Palette Read Index Register at I/O address 3C7h (or memory address offset 3C7h) and the Palette Write Index Register at I/O address 3C8h (or memory address offset 3C8h) are the two index registers. The Palette Read Index Register is the index register that is used to choose the color data position that is to be read from via the data port, while the Palette Write Index Register is the index register that is used to choose the color data position that is to be written to through the same data port. This arrangement allows the same data port to be used for reading from and writing to two different color data positions. Reading and writing the color data at a color data position involves three successive reads or writes since the color data stored at each color data position consists of three bytes.

To read a palette color data position, the index of the desired color data position must first be written to the Palette Read Index Register. Then all three bytes of data in a given color data position may be read at the Palette Data Register. The first byte read from the Palette Data Register retrieves the 8-bit value specifying the intensity of the red color component. The second and third bytes read are the corresponding 8-bit values for the green and blue color components respectively. After completing the third read operation, the Palette Read Index Register is automatically incremented so that the data of the next color data position becomes accessible for being read. This allows the contents of all of the 256 color data positions of the palette to be read in sequence. This is done by specifying only the index of the 0th color data position in the Palette Read Index Register, and then simply performing 768 successive reads from the Palette Data Register.

Writing a color data position, entails a very similar procedure. The index of the desired color data position must first be written to the Palette Write Index Register. Then all three bytes of data to specify a given color may be written to the Palette Data Register. The first byte written to the Palette Data Register specifies the intensity of the red color component, the second byte specifies the intensity for the green color component, and the third byte specifies the same for the blue color component. One important detail is that all three of these bytes must be written before the hardware will actually update these three values in the given color data position. When all three bytes have been written, the Palette Write Index Register is automatically incremented so that the data of the next color data position becomes accessible for being written. This allows the contents of all of the 256 color data positions of the palette to be written in sequence. This is done by specifying only the index of the 0th color data position in the Palette Write Index Register, and then simply performing 768 successive writes to the Palette Data Register.

DACMASK - Pixel Data Mask Register

I/O (and Memory Offset) Address: 3C6h

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Pixel Data Mask.** In indexed-color mode, the 8 bits of this register are logically ANDed with the 8 bits of pixel data received from the frame buffer for each pixel. The result of this ANDing process becomes the actual index used to select color data positions within the palette. This has the effect of limiting the choice of color data positions that may be specified by the incoming 8-bit data. 0 = Corresponding bit in the resulting 8-bit index being forced to 0. 1 = Allows the corresponding bit in the resulting index to reflect the actual value of the corresponding bit in the incoming 8-bit pixel data.

DACSTATE - DAC State Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Read Only

Bit	Description
7:2	**Reserved.** Read as 0.
1:0	**DACState.** This field indicates which of the two index registers was most recently written. **Bits [1:0]** Index Register Indicated 00 = Palette Write Index Register at I/O Address 3C7h (default) 01 = Reserved 10 = Reserved 11 = Palette Read Index Register at I/O Address 3C8h

DACRX - Palette Read Index Register

I/O (and Memory Offset) Address: 3C7h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Read Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being read from via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been read. A write to this register will abort a uncompleted palette write sequence. This register allows access to the palette even when running non-VGA display modes.

DACWX - Palette Write Index Register

I/O (and Memory Offset) Address: 3C8h

Default: 00h

Attributes: Write Only

Bit	Description
7:0	**Palette Write Index.** The 8-bit index value programmed into this register chooses which of 256 standard color data positions within the palette are to be made accessible for being written via the Palette Data Register (DACDATA). The index value held in this register is automatically incremented when all three bytes of the color data position selected by the current index have been written. This register allows access to the palette even when running non-VGA display modes.

DACDATA - Palette Data Register

I/O (and Memory Offset) Address: 3C9h

Default: Undefined

Attributes: Read/Write

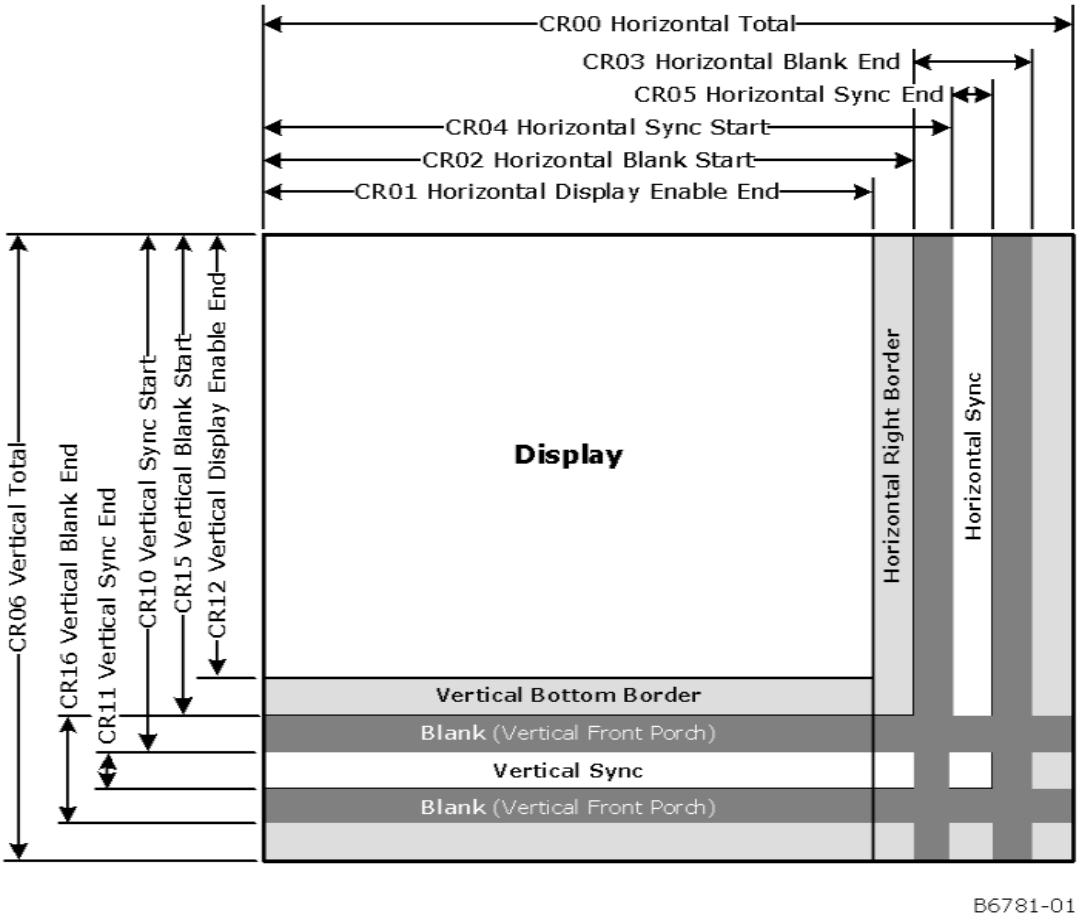
Bit	Description
7:0	**Palette Data.** This byte-wide data port provides read or write access to the three bytes of data of each color data position selected using the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX). The three bytes in each color data position are read or written in three successive read or write operations. The first byte read or written specifies the intensity of the red component of the color specified in the selected color data position. The second byte is for the green component, and the third byte is for the blue component. When writing data to a color data position, all three bytes must be written before the hardware will actually update the three bytes of the selected color data position. When reading or writing to a color data position, ensure that neither the Palette Read Index Register (DACRX) or the Palette Write Index Register (DACWX) are written to before all three bytes are read or written. A write to either of these two registers causes the circuitry that automatically cycles through providing access to the bytes for red, green and blue components to be reset such that the byte for the red component is the one that will be accessed by the next read or write operation via this register. This register allows access to the palette even when running non-VGA display modes. Writes to the palette can cause sparkle if not done during inactive video periods. This sparkle is caused by an attempt to write and read the same address on the same cycle. Anti-sparkle circuits will substitute the previous pixel value for the read output.

CRT Controller Register

For native VGA modes, the CRTC registers determine the display timing that is to be used. In centered VGA modes, these registers determine the size of the VGA image that is to be centered in the larger timing generator defined rectangle.

The CRT controller registers are accessed by writing the index of the desired register into the CRT Controller Index Register at I/O address 3B4h or 3D4h, depending on whether the graphics system is configured for MDA or CGA emulation. The desired register is then accessed through the data port for the CRT controller registers located at I/O address 3B5h or 3D5h, again depending upon the choice of MDA or CGA emulation as per MSR[0]. For memory mapped accesses, the Index register is at 3B4h (MDA mode) or 3D3h (CGA mode) and the data port is accessed at 3B5h (MDA mode) or 3D5h (CGA mode).

The following figure shows display fields and dimensions and the particular CRxx register that provides the control.



Group 0 Protection: In the original VGA, CR[0:7] could be made write-protected by CR11[7]. In BIOS code, this write protection is set following each mode change. Other protection groups have no current use, and would not be used going forward by the BIOS or by drivers. They are the result of an industry fad some years ago to attempt to write protect other groups of registers; however, all such schemes were chip specific. Only the write protection (Group 0 Protection) is supported.

CRX - CRT Controller Index Register

I/O (and Memory Offset) Address: 3B4h/3D4h

Default: 0Uh (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:0	**CRT Controller Index.** These 7 bits are used to select any one of the CRT controller registers to be accessed via the data port at I/O location 3B5h or 3D5h, depending upon whether the graphics system is configured for MDA or CGA emulation. The data port memory address offsets are 3B5h/3D5h.

CR00 - Horizontal Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=00h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Total.** This register is used to specify the total length of each scan line. This encompasses both the part of the scan line that is within the active display area and the part that is outside of it. Programming this register to a zero has the effect of stopping the fetching of display data. This field should be programmed with a value equal to the total number of character clocks within the entire length of a scan line, minus 5.

CR01 - Horizontal Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=01h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Display Enable End.** This register is used to specify the end of the part of the scan line that is within the active display area relative to its beginning. In other words, this is the horizontal width of the active display area. This field should be programmed with a value equal to the number of character clocks that occur within the horizontal active display area, minus 1. Horizontal display enable will go active at the beginning of each line during vertical active area, it will go inactive based on the programming of this register or the programming of the horizontal total (CR00) register. When this register value is programmed to a number that is larger than the total number of characters on a line, display enable will be active for all but the last character of the horizontal display line.

CR02 - Horizontal Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=02h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Blanking Start.** This register is used to specify the beginning of the horizontal blanking period relative to the beginning of the active display area of a scan line. Horizontal blanking should always be set to start no sooner than after the end of horizontal active. This field should be programmed with a value equal to the number of character clocks that occur on a scan line from the beginning of the active display area to the beginning of the horizontal blanking.

CR03 - Horizontal Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=03h)

Default: 1UUU UUUUb (U=Undefined)

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Reserved.** Values written to this bit are ignored, and to maintain consistency with the VGA standard, a value of 1 is returned when this bit is read. At one time, this bit was used to enable access to certain light pen registers. At that time, setting this bit to 0 provided this access, but setting this bit to 1 was necessary for normal operation.
6:5	**Display Enable Skew Control.** Defines the degree to which the start and end of the active display area are delayed along the length of a scan line to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = no delay 01 = delayed by 1 character clock 10 = delayed by 2 character clocks 11 = delayed by 3 character clocks
4:0	**Horizontal Blanking End Bits [4:0].** This field provides the 5 least significant bits of a 6-bit value that specifies the end of the blanking period relative to its beginning on a single scan line. Bit 7 of the Horizontal Sync End Register (CR05) supplies the most significant bit. This 6-bit value should be programmed to be equal to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02). End of blanking should occur before horizontal total.

CR04 - Horizontal Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=04h)

Default: Undefined

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Horizontal Sync Start** This register is used to specify the position of the beginning of the horizontal sync pulse relative to the start of the active display area on a scan line. This field should be set equal to the number of character clocks that occur from beginning of the active display area to the beginning of the horizontal sync pulse on a single scan line. Horizontal sync should always occur at least 2 clocks after the start of horizontal blank and 2 clocks before the end of horizontal blank. The actual start of sync will also be affected by both the horizontal sync skew register field and whether it is a text or graphics mode.

CR05 - Horizontal Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=05h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7	**Horizontal Blanking End Bit 5.** This bit provides the most significant bit of a 6-bit value that specifies the end of the horizontal blanking period relative to its beginning. Bits [4:0] of Horizontal Blanking End Register (CR03) supplies the 5 least significant bits. See CR03[4:0] for further details. This 6-bit value should be set to the least significant 6 bits of the result of adding the length of the blanking period in terms of character clocks to the value specified in the Horizontal Blanking Start Register (CR02).
6:5	**Horizontal Sync Delay.** This field defines the degree to which the start and end of the horizontal sync pulse are delayed to compensate for internal pipeline delays. This capability is supplied to implement VGA compatibility. These field describes the delay in terms of a number character clocks. **Bit [6:5] — Amount of Delay** 00 — no delay 01 — delayed by 1 character clock 10 — delayed by 2 character clocks 11 — delayed by 3 character clocks
4:0	**Horizontal Sync End.** This field provides the 5 least significant bits of a 5-bit value that specifies the end of the horizontal sync pulse relative to its beginning. A value equal to the 5 least significant bits of the horizontal character counter value at which time the horizontal retrace signal becomes inactive (logical 0). Thus, this 5-bit value specifies the width of the horizontal sync pulse. To obtain a retrace signal of W, the following algorithm is used: Value of Horizontal Sync start Register (CR04) + width of horizontal retrace signal in character clock units = 5 bit result to be programmed in this field

CR06 - Vertical Total Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=06h)

Default: 00h

Attributes: Read/Write (Group 0 Protection)

Bit	Description
7:0	**Vertical Total Bits [7:0].** This field provides the 8 least significant bits of either a 10-bit or 12-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. In standard VGA modes, the vertical total is specified with a 10-bit value. The 8 least significant bits of this value are supplied by these 8 bits of this register, and the 2 most significant bits are supplied by bits 5 and 0 of the Overflow Register (CR07).

CR07 - Overflow Register (Vertical)

I/O (and Memory Offset) Address: 3B5h/3D5h (index=07h)

Default: UU0U UUU0b (U=Undefined)

Attributes: Read/Write (Group 0 Protection on bits [7:5, 3:0])

Bit	Description
7	**Vertical Sync Start Bit 9.** The vertical sync start is a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by this bit and bit 2, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
6	**Vertical Display Enable End Bit 9.** The vertical display enable end is a 10-bit that specifies the number of the last scan line within the active display area. In standard VGA modes, the vertical display enable end is specified with a 10-bit value. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the most and second-most significant bits are supplied by this bit and bit 1, respectively, of this register. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
5	**Vertical Total Bit 9.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by this bit and bit 0, respectively, of this register. This 10-bit value should be programmed equal to the total number of scan lines, minus 2.
4	**Line Compare Bit 8.** This bit provides the second most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the

Bit	Description
	Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display what data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display what data exists in the frame buffer starting at the first byte of the frame buffer.
3	**Vertical Blanking Start Bit 8.** The vertical blanking start is a 10-bit that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by bit 5 of the Maximum Scan Line Register (CR09) and this bit of this register, respectively. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
2	**Vertical Sync Start Bit 8.** The vertical sync start is a 10-bit value that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Sync Start Register (CR10), and the most and second-most significant bits are supplied by bit 7 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the number of scan lines from the beginning of the active display area to the start of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.
1	**Vertical Display Enable End Bit 8.** The vertical display enable end is a 10-bit value that specifies the number of the last scan line within the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Display Enable End Register (CR12), and the two most significant bits are supplied by bit 6 and this bit, respectively, of this register. This 10-bit or value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area, minus 1.
0	**Vertical Total Bit 8.** The vertical total is a 10-bit value that specifies the total number of scan lines. This includes the scan lines both inside and outside of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Total Register (CR06), and the most and second-most significant bits are supplied by bit 5 and this bit, respectively, of this register. This 10-bit value should be programmed to be equal to the total number of scan lines, minus 2.

CR08 - Preset Row Scan Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=08h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0s.
6:5	**Byte Panning.** This field holds a 2-bit value that selects number of bytes (up to 3) by which the image is shifted horizontally to the left on the screen. This function is available in both text and graphics modes. In text modes with a 9-pixel wide character box, the image can be shifted up to 27 pixels to the left, in increments of 9 pixels. In text modes with an 8-pixel wide character box, and in all standard VGA graphics modes, the image can be shifted up to 24 pixels to the left, in increments of 8 pixels. When the Nine dot disable bit of the VGA control register is set, the pixel shift will be equivalent to the 8-dot mode. The image can be shifted still further, in increments of individual pixels, through the use of bits [3:0] of the Horizontal Pixel Panning Register (AR13).
4:0	**Starting Row Scan Count.** This field specifies which horizontal line of pixels within the character boxes of the characters used on the top-most row of text on the display will be used as the top-most scan line. The horizontal lines of pixels of a character box are numbered from top to bottom, with the top-most line of pixels being number 0. If a horizontal line of the these character boxes other than the top-most line is specified, then the horizontal lines of the character box above the specified line of the character box will not be displayed as part of the top-most row of text characters on the display. Normally, the value specified by these 5 bits should be 0, so that all of the horizontal lines of pixels within these character boxes will be displayed in the top-most row of text, ensuring that the characters in the top-most row of text do not look as though they have been cut off at the top.

Number of Pixels Shifted

Bit [6:5]	9-Pixel Text	8-Pixel Text & Graphics
00	0	0
01	9	8
10	18	16
11	27	24

CR09 - Maximum Scan Line Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=09h)

Default: 00h

Attributes: Read/Write

Bit	Description
7	**Double Scanning Enable.** 0 = Disable. When disabled, the clock to the row scan counter is equal to the horizontal scan rate. This is the normal setting for many of the standard VGA modes. 1 = Enable. When enabled, the clock to the row scan counter is divided by 2. This is normally used to allow CGA-compatible modes that have only 200 scan lines of active video data to be displayed as 400 scan lines (each scan line is displayed twice).
6	**Line Compare Bit 9.** This bit provides the most significant bit of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 4 of the Overflow Register (CR07) supplies the second most significant bit, and bits 7-0 of the Line Compare Register (CR18) supply the 8 least significant bits. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.
5	**Vertical Blanking Start Bit 9.** The vertical blanking start is a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area. The 8 least significant bits of this value are supplied by bits [7:0] of the Vertical Blanking Start Register (CR15), and the most and second-most significant bits are supplied by this bit and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal to the number of scan line from the beginning of the active display area to the beginning of the blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical blanking period begins.
4:0	**Starting Row Scan Count.** This field provides all 5 bits of a 5-bit value that specifies the number of scan lines in a horizontal row of text. This value should be programmed to be equal to the number of scan lines in a horizontal row of text, minus 1.

CR0A - Text Cursor Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ah)

Default: 00UU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7:6	**Reserved.** Read as 0.
5	**Text Cursor Off.** This text cursor exists only in text modes, so this register is entirely ignored in graphics modes. 0 = Enables the text cursor. 1 = Disables the text cursor.
4:0	**Text Cursor Start.** This field specifies which horizontal line of pixels in a character box is to be used to display the first horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the first horizontal line of pixels on which the cursor is to be shown.

CR0B - Text Cursor End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Bh)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6:5	**Text Cursor Skew.** This field specifies the degree to which the start and end of each horizontal line of pixels making up the cursor is delayed to compensate for internal pipeline delays. These 2 bits describe the delay in terms of a number character clocks. **Bit [6:5]** Amount of Delay 00 = No delay 01 = Delayed by 1 character clock 10 = Delayed by 2 character clocks 11 = Delayed by 3 character clocks
4:0	**Text Cursor End.** This field specifies which horizontal line of pixels in a character box is to be used to display the last horizontal line of the cursor in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the last horizontal line of pixels on which the cursor is to be shown.

CR0C - Start Address High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Ch)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [15:8].**This register provides either bits 15 through 8 of a 16-bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes, the start address is specified with a 16-bit value. The eight bits of this register provide the eight most significant bits of this value, while the eight bits of the Start Address Low Register (CR0D) provide the eight least significant bits.

CR0D - Start Address Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Dh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Start Address Bits [7:0]** This register provides either bits 7 through 0 of a 16 bit value that specifies the memory address offset from the beginning of the frame buffer at which the data to be shown in the active display area begins. (default is 0) In standard VGA modes the start address is specified with a 16-bit value. The eight bits of the Start Address High Register (CR0C) provide the eight most significant bits of this value, while the eight bits of this register provide the eight least significant bits.

CR0E - Text Cursor Location High Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Eh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [15:8].** This field provides the 8 most significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bit 7:0 of the Text Cursor Location Low Register (CR0F) provide the 8 least significant bits.

CR0F - Text Cursor Location Low Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=0Fh)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Text Cursor Location Bits [7:0].** This field provides the 8 least significant bits of a 16-bit value that specifies the address offset from the beginning of the frame buffer at which the text cursor is located. Bits 7:0 of the Text Cursor Location High Register (CR0E) provide the 8 most significant bits.

CR10 - Vertical Sync Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=10h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Sync Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit that specifies the beginning of the vertical sync pulse relative to the beginning of the active display area of a screen. In standard VGA modes, this value is described in 10 bits with bits [7,2] of the Overflow Register (CR07) supplying the 2 most significant bits. This 10-bit value should equal the vertical sync start in terms of the number of scan lines from the beginning of the active display area to the beginning of the vertical sync pulse. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which the vertical sync pulse begins.

CR11 - Vertical Sync End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=11h)

Default: 0U00 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Protect Registers [0:7].** The ability to write to Bit 4 of the Overflow Register (CR07) is not affected by this bit (i.e., bit 4 of the Overflow Register is always writeable). 0 = Enable writes to registers CR[00:07]. (default) 1 = Disable writes to registers CR[00:07].
6	**Reserved.** In the VGA standard, this bit was used to switch between 3 and 5 frame buffer refresh cycles during the time required to draw each horizontal line.
5	**Vertical Interrupt Enable.** This bit is reserved for compatibility only. While this bit may be written or read, it's value will have no effect. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. Bit 7 of Input Status Register 0 (ST00) originally indicated the status of the vertical retrace interrupt. 0 = Enable the generation of an interrupt at the beginning of each vertical retrace period. 1 = Disable the generation of an interrupt at the beginning of each vertical retrace period.
4	**Vertical Interrupt Clear.** This is reserved for compatibility only. VGA does not provide an interrupt signal which would be connected to an input of the system's interrupt controller. 0 = Setting this bit to 0 clears a pending vertical retrace interrupt. This bit must be set back to 1 to enable the generation of another vertical retrace interrupt.
3:0	**Vertical Sync End.** This 4-bit field provides a 4-bit value that specifies the end of the vertical sync pulse relative to its beginning. This 4-bit value should be set to the least significant 4 bits of the result of adding the length of the vertical sync pulse in terms of the number of scan lines that occur within the length of the vertical sync pulse to the value that specifies the beginning of the vertical sync pulse (see the description of the Vertical Sync Start Register for more details).

CR12 - Vertical Display Enable End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=12h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Display Enable End Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the number of the last scan line within the active display area. In standard VGA modes, this value is described in 10 bits with bits [6,1] of the Overflow Register (CR07) supplying the two most significant bits. This 10-bit value should be programmed to be equal to the number of the last scan line within in the active display area. Since the active display area always starts on the 0th scan line, this number should be equal to the total number of scan lines within the active display area,minus 1.

CR13 - Offset Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=13h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Offset Bits [7:0].** This register provides either all 8 bits of an 8-bit value that specifies the number of words or DWords of frame buffer memory occupied by each horizontal row of characters. Whether this value is interpreted as the number of words or DWords is determined by the settings of the bits in the Clocking Mode Register (SR01). In standard VGA modes, the offset is described with an 8-bit value, all the bits of which are provided by this register. This 8-bit value should be programmed to be equal to either the number of words or DWords (depending on the setting of the bits in the Clocking Mode Register, SR01) of frame buffer memory that is occupied by each horizontal row of characters.

CR14 - Underline Location Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=14h)

Default: 0UUU UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**Reserved.** Read as 0.
6	**DWord Mode.** 0 = Frame buffer addresses are interpreted by the frame buffer address decoder as being either byte addresses or word addresses, depending on the setting of bit 6 of the CRT Mode Control Register (CR17). 1 = Frame buffer addresses are interpreted by the frame buffer address decoder as being DWord addresses, regardless of the setting of bit 6 of the CRT Mode Control Register (CR17). This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to select how frame buffer addresses from the CPU are interpreted by the frame buffer address decoder as shown below: **CR14[6] / CR17[6] / Addressing Mode** 0 / 0 / Word Mode 0 / 1 / Byte Mode 1 / 0 / DWord Mode 1 / 1 / DWord Mode
5	**Count By 4.** 0 = The memory address counter is incremented either every character clock or every other character clock, depending upon the setting of bit 3 of the CRT Mode Control Register. 1 = The memory address counter is incremented either every 4 character clocks or every 2 character clocks, depending upon the setting of bit 3 of the CRT Mode Control Register. . This is used in mode x13 to allow for using all four planes. This bit is used in conjunction with bit 3 of the CRT Mode Control Register (CR17) to select the number of character clocks are required to cause the memory address counter to be incremented as shown, below: **CR14[5] / CR17[3] / Addressing Incrementing Interval** 0 / 0 / every character clock 0 / 1 / every 2 character clocks 1 / 0 / every 4 character clocks 1 / 1 / every 2 character clocks
4:0	**Underline Location.** This field specifies which horizontal line of pixels in a character box is to be used to display a character underline in text mode. The horizontal lines of pixels in a character box are numbered from top to bottom, with the top-most line being number 0. The value specified by these 5 bits should be the number of the horizontal line on which the character underline mark is to be shown.

CR15 - Vertical Blanking Start Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=15h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Vertical Blanking Start Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the beginning of the vertical blanking period relative to the beginning of the active display area of the screen. In standard VGA modes, the vertical blanking start is specified with a 10-bit value. The most and second-most significant bits of this value are supplied by bit 5 of the Maximum Scan Line Register (CR09) and bit 3 of the Overflow Register (CR07), respectively. This 10-bit value should be programmed to be equal the number of scan lines from the beginning of the active display area to the beginning of the vertical blanking period. Since the active display area always starts on the 0th scan line, this number should be equal to the number of the scan line on which vertical blanking begins.

CR16 - Vertical Blanking End Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=16h)

Default: Undefined

Attributes: Read/Write

This register provides a 8-bit value that specifies the end of the vertical blanking period relative to its beginning.

Bit	Description
7:0	**Vertical Blanking End Bits [7:0].** This 8-bit value should be set equal to the least significant 8 bits of the result of adding the length of the vertical blanking period in terms of the number of scan lines that occur within the length of the vertical blanking period to the value that specifies the beginning of the vertical blanking period (see the description of the Vertical Blanking Start Register for details).

CR17 - CRT Mode Control

I/O (and Memory Offset) Address: 3B5h/3D5h (index=17h)

Default: 0UU0 UUUUb (U=Undefined)

Attributes: Read/Write

Bit	Description
7	**CRT Controller Reset.** This bit has no effect except in native VGA modes (non-centered). 0 = Forces horizontal and vertical sync signals to be inactive. No other registers or outputs are affected. 1 = Permits normal operation.
6	**Word Mode or Byte Mode.** 0 = The memory address counter's output bits are shifted by 1 bit position before being passed on to the frame buffer address decoder such that they are made into word-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. 1 = The memory address counter's output bits remain unshifted before being passed on to the frame buffer address decoder such that they remain byte-aligned addresses when bit 6 of the Underline Location Register (CR17) is set to 0. This bit is used in conjunction with bits 6 and 5 of the CRT Mode Control Register (CR17) to control how frame buffer addresses from the memory address counter are interpreted by the frame buffer address decoder as shown below: CR14[6] CR17[6] **Address Mode** 0 0 Word Mode - Addresses from the memory address counter are shifted once to become word-aligned 0 1 Byte Mode - Addresses from the memory address counter are not shifted 1 0 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned 1 1 DWord Mode - Addresses from the memory address counter are shifted twice to become DWord-aligned
5	**Address Wrap.** This bit is only effective when word mode is made active by setting bit 6 in both the Underline Location Register and this register to 0. 0 = Wrap frame buffer address at 16 KB. This is used in CGA-compatible modes. 1 = No wrapping of frame buffer addresses.
4	**Reserved.** Read as 0.

Bit	Description
3	**Count By 2.** This bit is used in conjunction with bit 5 of the Underline Location Register (CR14) to select the number of character clocks are required to cause the memory address counter to be incremented. 0 = The memory address counter is incremented either every character clock or every 4 character clocks, depending upon the setting of bit 5 of the Underline Location Register. 1 = The memory address counter is incremented either every other clock. CR14[5] CR17[3] Address Incrementing interval 0 0 every character clock 0 1 every 2 character clocks 1 0 every 4 character clocks 1 1 every 2 character clocks
2	**Horizontal Retrace Select.** This bit provides a way of effectively doubling the vertical resolution by allowing the vertical timing counter to be clocked by the horizontal retrace clock divided by 2 (usually, it would be undivided). 0 = The vertical timing counter is clocked by the horizontal retrace clock. 1 = The vertical timing counter is clocked by the horizontal retrace clock divided by 2.
1	**Select Row Scan Counter.** 0 = A substitution takes place, where bit 14 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 1 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.
0	**Compatibility Mode Support.** 0 = A substitution takes place, where bit 13 of the 16-bit memory address generated of the memory address counter (after the stage at which these 16 bits may have already been shifted to accommodate word or DWord addressing) is replaced with bit 0 of the row scan counter at a stage just before this address is presented to the frame buffer address decoder. 1 = No substitution takes place. See following tables.

The following tables show the possible ways in which the address bits from the memory address counter can be shifted and/or reorganized before being presented to the frame buffer address decoder. First, the address bits generated by the memory address counter are reorganized, if need be, to accommodate byte, word or DWord modes. The resulting reorganized outputs (MAOut15-MAOut0) from the memory address counter may also be further manipulated with the substitution of bits from the row scan counter (RSOut1 and RSOut0) before finally being presented to the input bits of the frame buffer address decoder (FBIn15-FBIn0).

Memory Address Counter Address Bits [15:0]

	Byte Mode CR14 bit 6=0 CR17 bit 6=1 CR17 bit 5=X	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=1	Word Mode CR14 bit 6=0 CR17 bit 6=0 CR17 bit 5=0	DWord Mode CR14 bit 6=1 CR17 bit 6=X CR17 bit 5=X
MAOut0	0	15	13	12
MAOut1	1	0	0	13
MAOut2	2	1	1	0
MAOut3	3	2	2	1
MAOut4	4	3	3	2
MAOut5	5	4	4	3
MAOut6	6	5	5	4
MAOut7	7	6	6	5
MAOut8	8	7	7	6
MAOut9	9	8	8	7
MAOut10	10	9	9	8
MAOut11	11	10	10	9
MAOut12	12	11	11	10
MAOut13	13	12	12	11
MAOut14	14	13	13	12
MAOut15	15	14	14	13

X = Don't Care

Frame Buffer Address Decoder

	CR17 bit 1=1	CR17 bit 1=1	CR17 bit 1=0	CR17 bit 1=0
	CR17 bit 0=1	CR17 bit 0=0	CR17 bit 0=1	CR17 bit 0=0
FBIn0	MAOut0	MAOut0	MAOut0	MAOut0
FBIn1	MAOut1	MAOut1	MAOut1	MAOut1
FBIn2	MAOut2	MAOut2	MAOut2	MAOut2
FBIn3	MAOut3	MAOut3	MAOut3	MAOut3
FBIn4	MAOut4	MAOut4	MAOut4	MAOut4
FBIn5	MAOut5	MAOut5	MAOut5	MAOut5
FBIn6	MAOut6	MAOut6	MAOut6	MAOut6
FBIn7	MAOut7	MAOut7	MAOut7	MAOut7
FBIn8	MAOut8	MAOut8	MAOut8	MAOut8
FBIn9	MAOut9	MAOut9	MAOut9	MAOut9
FBIn10	MAOut10	MAOut10	MAOut10	MAOut10
FBIn11	MAOut11	MAOut11	MAOut11	MAOut11
FBIn12	MAOut12	MAOut12	MAOut12	MAOut12
FBIn13	MAOut13	MAOut13	RSOut0	RSOut0
FBIn14	MAOut14	RSOut1	MAOut14	RSOut1
FBIn15	MAOut15	MAOut15	MAOut15	MAOut15

CR18 - Line Compare Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=18h)

Default: Undefined

Attributes: Read/Write

Bit	Description
7:0	**Line Compare Bits [7:0].** This register provides the 8 least significant bits of a 10-bit value that specifies the scan line at which the memory address counter restarts at the value of 0. Bit 6 of the Maximum Scan Line Register (CR09) supplies the most significant bit, and bit 4 of the Overflow Register (CR07) supplies the second most significant bit. Normally, this 10-bit value is set to specify a scan line after the last scan line of the active display area. When this 10-bit value is set to specify a scan line within the active display area, it causes that scan line and all subsequent scan lines in the active display area to display video data starting at the very first byte of the frame buffer. The result is what appears to be a screen split into a top and bottom part, with the image in the top part being repeated in the bottom part. (This register is only used in split screening modes, and this is not a problem because split screening is not actually used for extended modes. As a result, there is no benefit to extending the existing overflow bits for higher resolutions.) When used in cooperation with the Start Address High Register (CR0C) and the Start Address Low Register (CR0D), it is possible to create a split display, as described earlier, but with the top and bottom parts displaying different data. The top part will display whatever data exists in the frame buffer starting at the address specified in the two aforementioned start address registers, while the bottom part will display whatever data exists in the frame buffer starting at the first byte of the frame buffer.

CR22 - Memory Read Latch Data Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=22h)

Default: 00h

Attributes: Read Only

Bit	Description
7:0	**Memory Read Latch Data.** This field provides the value currently stored in 1 of the four memory read latches. Bits 1 and 0 of the Read Map Select Register (GR04) select which of the four memory read latches may be read via this register.

CR24 - Toggle State of Attribute Controller Register

I/O (and Memory Offset) Address: 3B5h/3D5h (index=24h)

Default: 00h

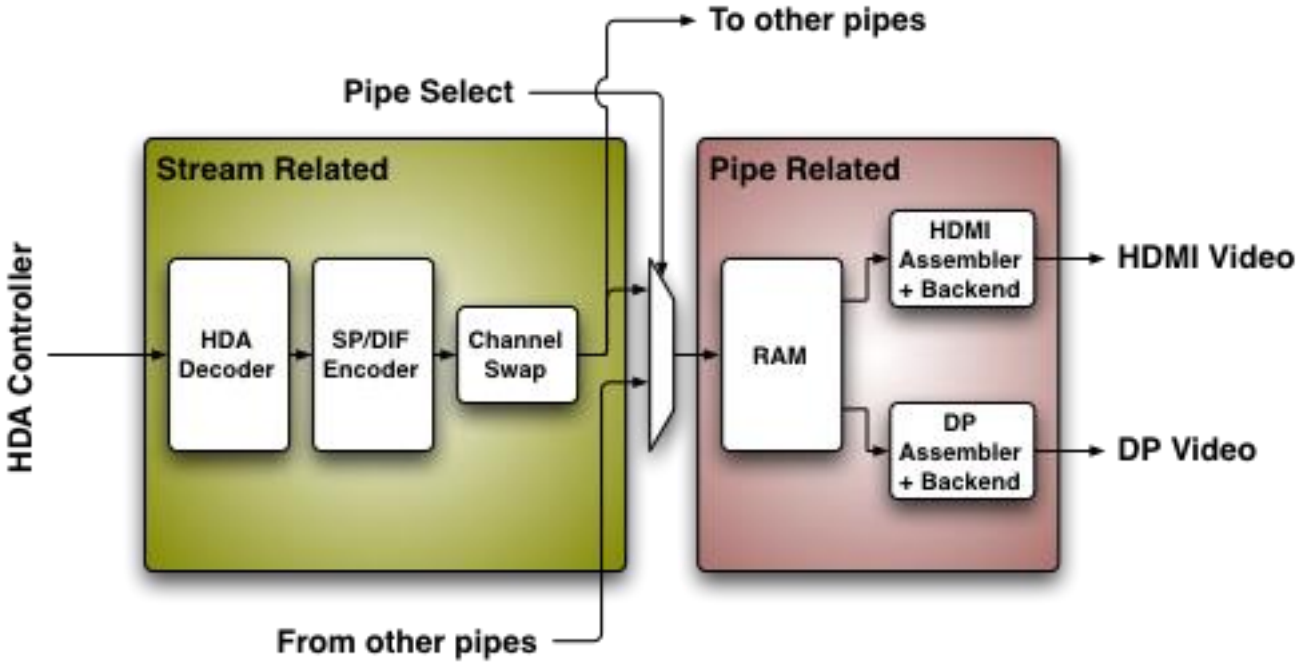
Attributes: Read Only

Bit	Description
7	**Toggle Status.** Indicates where the last write to attribute register was to: 0 = index port 1 = data port
6:0	**Reserved.** Read as 0.

Display Audio Codec Verbs

Content for this topic is currently under development.

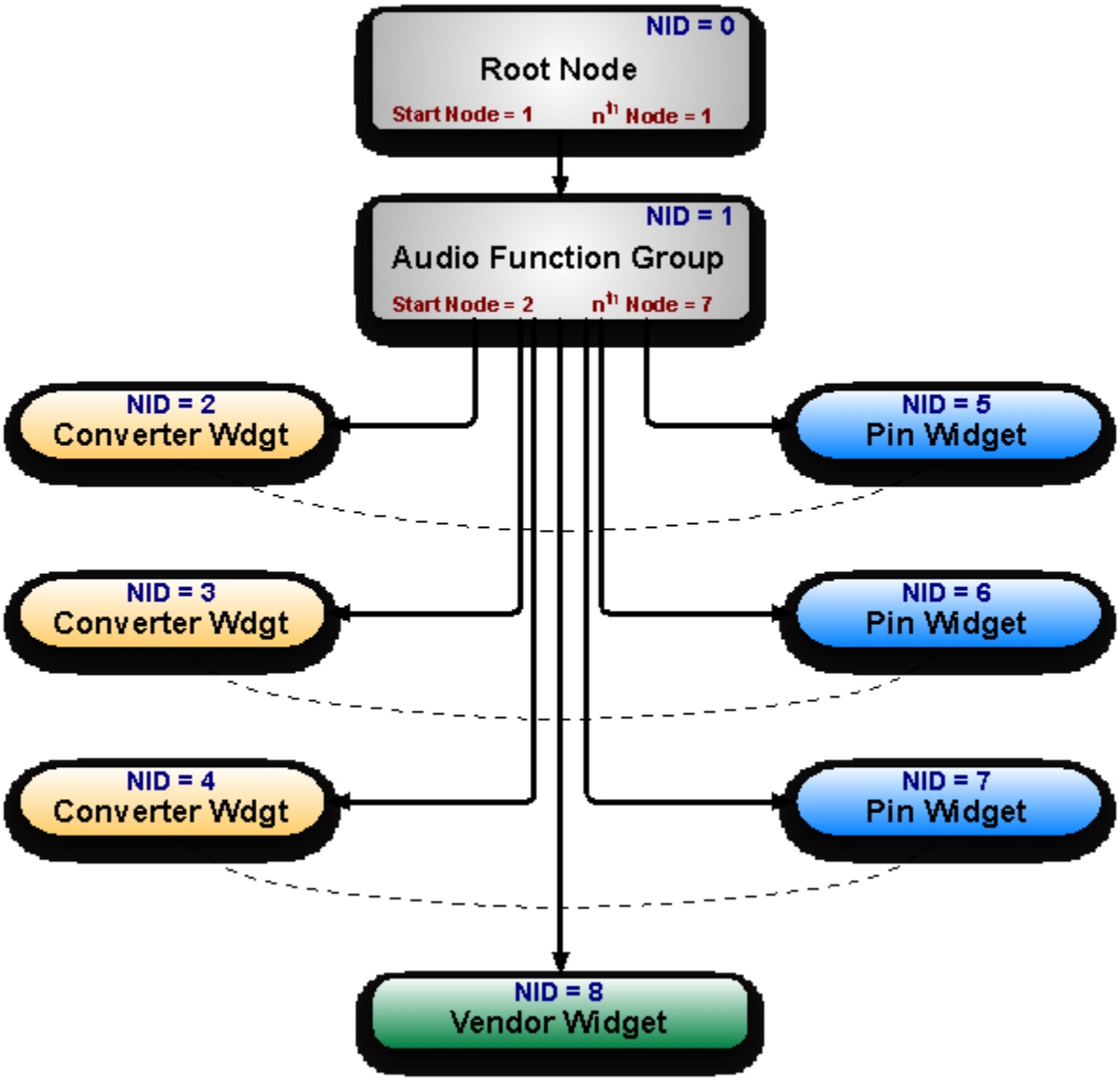
Block Diagram



Codec Node Hierarchy

The diagram below shows the hierarchy of the internal codec. The codec is presented as a single codec with multiple endpoints. By operating as a single codec, only one driver needs to be loaded on the system.

Inside the codec are three "converter widgets" and three "pin widgets", responsible for taking data from HD Audio DMA engines and placing into an HDMI/DP stream. Each pin widget has a 1-1 connection to a converter widget (as indicated by the dotted lines in the diagram).



Programming

Programming of the codec is performed by "verbs" as described in the HD Audio specification. These verbs travel over the internal HD Audio link at a rate of 1 verb per frame. A verb can either come from the CORB, with responses using the RIRB, or using an immediate command and response mechanism (ICR). Device 2 contains its own copy of an ICR mechanism as a back-door into the audio codec.

Verb Support

Verb ID			Node ID							
Set	Get	Verb Name/Description	01h	02h	03h	04h	05h	06h	07h	08h
2h	Ah	Stream Descriptor Format		Y	Y	Y				
3h	Bh	Set Amplifier Mute					Y	Y	Y	
-	F00h	Get Parameters	Y	Y	Y	Y	Y	Y	Y	Y
701h	F01h	Connection Select Control					Y	Y	Y	
-	F02h	Connection List Entry					Y	Y	Y	
705h	F05h	Power State		Y	Y	Y	Y	Y	Y	
706h	F06h	Channel and Stream ID		Y	Y	Y				
707h	F07h	Pin Widget Control					Y	Y	Y	
708h	F08h	Unsolicited Response Enable					Y	Y	Y	
-	F09h	Pin Sense					Y	Y	Y	
-	F0Dh	Digital Converter		Y	Y	Y				
70Dh	-	Digital Converter 1		Y	Y	Y				
70Eh	-	Digital Converter 2		Y	Y	Y				
-	F1Ch	Configuration Default					Y	Y	Y	
71Ch	-	Configuration Default Byte 0					Y	Y	Y	
71Dh	-	Configuration Default Byte 1					Y	Y	Y	
71Eh	-	Configuration Default Byte 2					Y	Y	Y	
71Fh	-	Configuration Default Byte 3					Y	Y	Y	
-	F20h	Subsystem ID	Y							
-	F21h	Subsystem ID	Y							
-	F22h	Subsystem ID	Y							
-	F23h	Subsystem ID	Y							
720h	-	Subsystem ID[7: 0]	Y							
721h	-	Subsystem ID[15: 8]	Y							
722h	-	Subsystem ID[23:16]	Y							
723h	-	Subsystem ID[31:24]	Y							
72Dh	F2Dh	Converter Channel Count		Y	Y	Y				
-	F2Eh	HDMI/DP Info Size					Y	Y	Y	

Verb ID			Node ID							
Set	Get	Verb Name/Description	01h	02h	03h	04h	05h	06h	07h	08h
730h	F30h	HDMI Info Index					Y	Y	Y	
731h	F31h	HDMI Info Data					Y	Y	Y	
732h	F32h	HDMI Info Transmit Control					Y	Y	Y	
734h	F34h	Converter Channel Map					Y	Y	Y	
735h	F35h	Device Select					Y	Y	Y	
-	F36h	Display Device List Entry					Y	Y	Y	
73Ch	73Ch	DisplayPort Stream ID					Y	Y	Y	
73Eh	-	Digital Converter 3		Y	Y	Y				
73Fh	-	Digital Converter 4		Y	Y	Y				
-	F80h	HDMI / DP Status								Y
781h	F81h	HDMI Vendor Verb								Y
		delete?								Y
-	F83h	Captured Wall Clock Value								Y
-	F84h	Captured GTC Value								Y
-	F85h	Get GTC Offset Value								Y
785h	-	Set GTC Offset Value[7: 0]								Y
786h	-	Set GTC Offset Value[15: 8]								Y
787h	-	Set GTC Offset Value[23:16]								Y
788h	-	Set GTC Offset Value[31:24]								Y
789h	F89h	Converter Channel Count								Y

Parameter Support

Param ID	Parameter Name	Node ID								
		00h	01h	02h	03h	04h	05h	06h	07h	08h
00h	Vendor ID	Y								
02h	Revision ID	Y								
04h	Subordinate Node Count	Y	Y							
05h	Function Group Type		Y							
08h	Audio Function Group Capabilities									
09h	Audio Widget Capabilities			Y	Y	Y	Y	Y	Y	Y
0Ah	Sample Size, Rate CAPs			Y	Y	Y				
0Bh	Stream Formats			Y	Y	Y				
0Ch	Pin Capabilities						Y	Y	Y	
0Dh	Input Amp Capabilities									
0Eh	Connection List Length						Y	Y	Y	
0Fh	Supported Power States		Y							
10h	Processing Capabilities									
11h	GPIO Count									
12h	Output Amp Capabilities						Y	Y	Y	
13h	Volume Knob Capabilities									
15h	Device List Length						Y	Y	Y	

Node ID 00h: Root Node Verbs

The root node only contains a single verb - the "Get Parameters" verb at F00h.

F00h - Get Parameters

Parameter	Symbol	Register Name
00h	PARAM_VID	Vendor ID
02h	PARAM_RID	Revision ID
04h	PARAM_SNC	Subordinate Node Count

Parameter 00h: VID - Vendor ID

Bit	Reset	Description
31:16	8086h	**Vendor ID (VID):** Indicates the 16-bit Vendor ID values used to identify the codec to the PnP subsystem.
15:00	2808h	**Device ID (DID):** Indicates the 16-bit Device ID values used to identify the codec to the PnP subsystem.

Parameter 02h: RID - Revision ID

Bit	Reset	Description
31:24	*0*	*Reserved*
23:20	1h	**Major Revision (MJR):** Indicates the major revision number (left of the decimal) of the High Definition Audio Specification to which the codec is fully compliant.
19:16	0h	**Minor Revision (MNR):** Indicates the minor revision number (right of the decimal) or "dot number" of the High Definition Audio Specification to which the codec is fully compliant.
15:08	00h	**Revision ID (RID):** Indicates the vendor's revision number for this given Device ID.
07:00	00h	**Stepping ID (SID):** Indicates optional vendor stepping number within the revision.

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	0h	**Starting Node Number (SNN):** Indicates the first sub-node's ID is 01h.
15:08	00h	Reserved
07:00	01h	**Total Number of Nodes (TNN):** Indicates one sub-node

Node ID 01h: Audio Function Group Verbs

Set Verb	Get Verb	Symbol	Name
-	F00h	GET PARAM	Get Parameters
705h	F05h	SET_PS / GET_PS	Set Power State
-	F20h	GET_SSID	Get Subsystem ID
720h	720h	SET_SSID0	Set Subsystem ID
721h	721h	SET_SSID1	Set Subsystem ID
722h	722h	SET_SSID2	Set Subsystem ID
723h	723h	SET_SSID3	Set Subsystem ID

F00h: Get Parameters

Parameter	Symbol	Register Name
04h	PARAM_SNC	Subordinate Node Count
05h	PARAM_FGT	Function Group Type
08h	PARAM FGC	Function Group Capabilities
0Fh	PARAM SPS	Supported Power States

Parameter 04h: PARAM_SNC - Subordinate Node Count

Bit	Reset	Description
31:24	*0*	*Reserved*
23:16	02h	**Start Node Number (SNN):** Indicates the start node number of widget or functional nodes in the Functional Group.
15:08	*0*	*Reserved*
07:00	07h	**Total Number of Nodes (TNN):** Indicates 7 widgets in the Functional Group. (HDMI/DP converters (3) + HDMI/DP pins (3) + Vendor Defined Widget (1))

Parameter 05h: PARAM_FGT - Function Group Type

Bit	Reset	Description
31:09	*0*	*Reserved*
08	0	**Unsolicited Capable (UC):** Not capable of generating an unsolicited response.
07:00	01h	**Node Type (NT):** Indicates Audio Function Group.

Parameter 08h: PARAM_FGC - Function Group Capability

Bit	Reset	Description
31:04	*0*	*Reserved*
03:00	00h	**Output Delay (OD)**Output Delay.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30	1	**Clock Stop (CS):** Indicates support for D3 when clock is stopped.
29:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

705h: SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h: GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	1	**Clock Stop OK (CSOK):** Clock stopping in D3 is OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

F20h: GET SSID - Get Subsystem ID0

Bits	Reset	Description
31:00	80860101h	**Subsystem ID (SSID):** Reports the sub-system ID set via SET_SSIDx verbs.

720h: SET SSID0 - Set Subsystem ID0

Bits	Description
07:00	Subsystem ID Bits [7:0]

721h: SET SSID1 - Set Subsystem ID1

Bits	Description
07:00	Subsystem ID Bits [15:8]

722h: SET SSID2 - Set Subsystem ID2

Bits	Description
07:00	Subsystem ID Bits [23:16]

723h: SET SSID3 - Set Subsystem ID3

Bits	Description
07:00	Subsystem ID Bits [31:24]

Node ID 02h, 03h, 04h: Audio Output Convertor Widget Verbs

Verb	Symbol	Verb Name
2h	SET_SDF	Set Stream Descriptor Format
Ah	GET_SDF	Get Stream Descriptor Format
F00h	GET_PARAM	Get Parameters
705h	SET_PS	Set Power State
F05h	GET_PS	Get Power State
706h	SET_CSID	Set Channel and Stream ID
F06h	GET_CSID	Get Channel and Stream ID
F0Dh	SET_DC1	Get Digital Converter
70Dh	SET_DC1	Set Digital Converter 1
70Eh	SET_DC2	Set Digital Converter 2
73Eh	SET_DC3	Set Digital Converter 3
73Fh	SET_DC4	Set Digital Converter 4
72Dh	SET_CCC	Set Converter Channel Count
F2Dh	GET_CCC	Get Converter Channel Count

2h/Ah: SET/GET_SDF - Set/GET Stream Descriptor Format

Bits	Reset	Description
31:15	0	Reserved
14	0	**Sample Base Rate (SBR):**
13:11	000	**Sample Base Rate Multiplier (SBRM):**
10:08	000	**Sample Base Rate Divisor (SBRD):**
07	0	Reserved
06:04	011	**Bits / Sample (BPS):** • 001b: Data is packed in memory in 16 bit containers on 16 bit boundaries • 010b: Data is packed in memory in 20 bit containers on 32 bit boundaries • 011b: Data is packed in memory in 24 bit containers on 24 bit boundaries • 100b: Data is packed in memory in 32 bit containers on 32 bit boundaries All other bit combinations reserved
03:00	1h	**# Channels in Stream (NCS):** 2 channels in each frame

F00h: Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ah	PARAM_PSB	Parameter Sizes and Bit Rates
0Bh	PARAM_SF	Stream Formats
0Fh	PARAM_SPS	Supported Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	0h	**Widget Type (TYPE):** Indicates this is an audio output widget
19:16		Sample Delay in Widget (DELAY):
15:13	011	**Channel Count Extension (CCE):** These three bits, combined with STRO, indicate that there are 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	0	**Connection List (CL):** Indicates no connection list
07	0	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	1	**Format Override (FO):** Indicates support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier support.
02	0	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ah: PSB - PCM Sizes and Bit Rates

Bits	Reset	Description
31:21	*0*	*Reserved*
20	1	**32-bit Support (B32):** Indicates 32-bit samples supported
19	1	**24-bit Support (B24):** Indicates 24-bit samples supported
18	1	**20-bit Support (B20):** Indicates 20-bit samples supported
17	1	**16-bit Support (B16):** Indicates 16-bit samples supported
16	0	**8-bit Support (B8):** Indicates 8-bit samples not supported

Bits	Reset	Description
15:12	*0*	*Reserved*
11	0	**384 kHz Support (R12):** Indicates 384 kHz not supported
10	1	**192 kHz Support (R11):** Indicates 192 kHz supported
09	1	**176.4 kHz Support (R10):** Indicates 176.4 kHz supported
08	1	**96 kHz Support (R9):** Indicates 96 kHz supported
07	1	**88.2 kHz Support (R8):** Indicates 88.2 kHz supported
06	1	**48 kHz Support (R7):** Indicates 48 kHz supported
05	1	**44.1 kHz Support (R6):** Indicates 44.1 kHz supported
04	1	**32 kHz Support (R5):** Indicates 32 kHz supported
03	0	**22.05 kHz Support (R4):** Indicates 22.05 kHz not supported
02	0	**16 kHz Support (R3):** Indicates 16 kHz not supported
01	0	**11.025 kHz Support (R2):** Indicates 11.025 kHz not supported
00	0	**8 kHz Support (R1):** Indicates 8 kHz not supported

Parameter 0Bh: SF - Stream Formats

Bits	Reset	Description
31:03	0	*Reserved*
02	1	**AC3 Support (AC3):** Indicates AC3 stream format is supported
01	0	**Float32 Support (F32):** Indicates float32 stream format not supported
00	1	**PCM Support (PCM):** Indicates PCM format is supported.

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

705h: SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h: GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** ???
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

706h/F06h: GET/SET_CSID - Get/Set Channel and Stream ID

Bits	Reset	Description
07:04	0h	**Stream ID (SID):** Link stream used by the converter for data output.
03:00	0h	**Lowest Channel Number (LCN):** Lowest channel used by the converter.

Digital Converter Verbs

F0Dh: GET_DC - Get Digital Converter

Bits	Reset	Description
31:24	0	Reserved
23	1	**Keep Alive (KA):** See SET_DC3.KA
22:20	0	Reserved
19:16	0h	**IEC Coding Type (ICT):** See SET_DC3.ICT
15	0	Reserved
14:08	00h	**Category Code (CC):** See SET_DC1.CC
07	0	**Level (LVL):** See SET_DC1.LVL
06	0	**Professional (PRO):** See SET_DC1.PRO
05	0	**Audio is not PCM (AUDIO):** See SET_DC1.AUDIO
04	0	**Copyright (COPY):** See SET_DC1.COPY
03	0	**Pre-emphasis (PRE):** See SET_DC1.PRE
02	0	**Validity Configuration (VCFG):** See SET_DC1.VCFG
01	0	**Validity (V):** See SET_DC1.V
00	1	**Digital Enable (DIGEN):** See SET_DC1.DIGEN

70Dh: SET_DC1 - Set Digital Converter 1

Bits	Description
07	**Level (LVL):** S/PDIF IEC Generation Level.
06	**Professional (PRO):** When set, indicates professional use of channel.
05	**Audio is not PCM (AUDIO):** When set, data is non-PCM format.
04	**Copyright (COPY):** When set, copyright asserted.
03	**Pre-emphasis (PRE):** When set, enables filter pre-emphasis.
02	**Validity Configuration (VCFG):** Determines S/PDIF transmitter behavior when data is not being transmitted.
01	**Validity (V):** Affects the validity flag transmitted in each sub-frame, and enables S/PDIF transmitter to maintain connection during error or mute conditions.
00	**Digital Enable (DIGEN):** When set, enables digital content

70Eh: Digital Converter 2

Bits	Description
07	Reserved
06:00	**Category Code (CC):** S/PDIF IEC Category Code.

73Eh: Digital Converter 3

Bits	Description
07	Keep Alive
06:04	Reserved
03:00	IEC Coding Type

73Fh: Digital Converter 4

Bits	Description
07:00	Reserved

72Dh/F2Dh: GET/SET_CCC - Get/Set Converter Channel Count

Bits	Reset	Description
07:04	0	Reserved
03:00	0000	Converter Channel Count 1 (0^{th} order)

Node ID 05h, 06h, 07h: Pin Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
3h	-	SET_AM	Set Amplifier Mute
-	Bh	GET_AM	Get Amplifier Mute
-	F00h	-	Get Parameters
701h	F01h	SET_CSC / GET_CSC	Set/Get Connection Select Control
-	F02h	-	Get Connection List Entry
705h	F05h	SET_PS / GET_PS	Set/Get Power State
707h	F07h	SET_PWC / GET_PWC	Set/Get Pin Widget Control
708h	F08h	SET_UE / GET_UE	Set/Get Unsolicited Response Enable
-	F09h	-	Get Pin Sense
71Ch	-	SET_CD0	Set Configuration Default Byte 0
71Dh	-	SET_CD1	Set Configuration Default Byte 1
71Eh	-	SET_CD2	Set Configuration Default Byte 2
71Fh	-	SET_CD3	Set Configuration Default Byte 3
-	F1Ch	GET_CD	Get Configuration Default
-	F2Eh	GET_HDIS	Get HDMI/DP Info Size
730h	F30h	SET_HII / GET_HII	Set/Get HDMI Info Index
731h	F31h	SET_HID / GET_HID	Set/Get HDMI Info Data
732h	F32h	SET_HITC / GET_HITC	Set/Get HDMI Info Transmit Control
733h	F33h	SET_PC / GET_PC	Set/Get Protection Control
734h	F34h	SET_CCM / GET_CCM	Set/Get Converter Channel Map
735h	F35h	SET_DS / GET_DS	Set/Get Device Select
-	F36h	GET_DDLE	Get Display Device List Entry
73Ch	F3Ch	SET_DPID / GET_DPID	Set/Get DisplayPort Stream ID

3h: SET_AM - Set Amplifier Mute

Bits	Bits	Description
15	0	**Set Output Amp (SOA):**.
14	0	**Set Input Amp (SIA):**.
13	0	**Set Left Amp (SLA):**.
12	0	**Set Right Amp (SRA):**.
11:08	0h	**Index (IDX):**
07	1	**Mute (MUTE):** When set, amp muted.
06:00	*0*	*Reserved*

B8h: GET_AM - Get Amplifier Mute

Bits	Bits	Description
31:08	*0*	*Reserved*
07	1	**Mute (MUTE):** When set, amp muted.
06:00	*0*	*Reserved*

F00h: Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities
0Ch	PARAM_PC	Pin Capabilities
0Eh	PARAM_CLL	Connection List Length
12h	PARAM OAC	Output Amplifier Capabilities
15h	PARAM_DLL	Device List Length
0Fh	PARAM_SPS	Supported Power States

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	4h	**Widget Type (TYPE):** Indicates this is a pin complex widget
19:16	0	**Sample Delay in Widget (DELAY):** No delay through the pin widget.
15:13	011	**Channel Count Extension (CCE):** This field, combined with STRO, indicate 8 channels supported.
11	0	**L-R Swap (LRS):** Indicates no left/right channel swap.
10	1	**Power Control (PC):** Indicates power state control
09	1	**Digital (DIG):** Indicates support for digital streams.
08	1	**Connection List (CL):** Indicates a connection list
07	1	**Unsolicited Capable (UC):** Indicates support for unsolicited responses.
06	0	**Processing Widget (PW):** Indicates no support for processing
05	0	**Stripe (STRP):** Indicates striping not supported.
04	0	**Format Override (FO):** Indicates no support for formatting
03	1	**Amp Parameter Override (APO):** Indicates no amplifier override support.
02	1	**Out Amp Present (OAP):** Indicates no output amplifier present.
01	0	**In Amp Present (IAP):** Indicates no input amplifier present.
00	1	**Stereo (STRO):** Indicates a stereo widget

Parameter 0Ch: PC - Pin Capabilities

Bits	Reset	Description
31:28	*0*	*Reserved*
27	1	**High Bit Rate (HBR):** Indicates support for high bit-rate audio
26	*0*	*Reserved*
25	1	**DisplayPort Multi Stream Capable(MSC):** Indicates support for DisplayPort Multistream. Will be 0 in vanilla mode.
24	1	**DisplayPort (DP):** Indicates support for DisplayPort
23:08	*0*	*Reserved*
07	1	**HDMI (HDMI):** Indicates support for HDMI
06:05	*0*	*Reserved*
04	1	**Output Capable (OC):** Pin is output capable
03	*0*	*Reserved*
02	1	**Presence Detect Capable (PDC):** Indicates capability for presence detection
01:00	*0*	*Reserved*

Parameter 0Eh: CLL - Connection List Length

Bits	Reset	Description
31:08	*0*	*Reserved*
07	0	**Long Form (LF):** Indicates connection list is short form
06:00	03h	**Length (LEN):** Indicates there is one item in the connection list.

Parameter 12h: OAC - Output Amplifier Capabilities

Bits	Reset	Description
31	1	**Mute Capable (MC):** Muting is capable on this pin
30:00	0	Reserved

Parameter 15h: DLL - Device List Length

Bits	Reset	Description
31:06	*0*	*Reserved*
05:00	00h	**Length (LEN):** Indicates no devices

Parameter 0Fh: PARAM_SPS - Supported Power States

Bit	Reset	Description
31	1	**Extended Power State Supported (EPSS):** Indicates support for low power states
30:04	0	*Reserved*
03	1	**D3 Supported (D3S):** Indicates support for D3.
02	0	**D2 Supported (D2S):** Indicates no support for D2.
01	0	**D1 Supported (D1S):** Indicates no support for D1.
00	1	**D0 Supported (D0S):** Indicates support for D0.

701h/F01h: SET/GET_CSC - Set/Get Connection Select Control

Bits	Reset	Description
07:00	00h	**Connection Select Control (CSC):**

F02h: GET_CLE - Get Connection List Entry

Bits	Reset	Description
31:08	0	Reserved
07:00	Varies	**Connection List Entry (CLE):** 02h for NodeID 05h, 03h for NodeID 06h, and 04h for NodeID 07h.

705h: SET_PS - Set Power State

Bits	Description
07:02	Reserved
01:00	**Requested Power State (RPS):** Only D0 (00) and D3 (11) may be requested

F05h: GET_PS - Get Power State

Bits	Reset	Description
31:11	0	Reserved
10	0	**Settings Reset (SR):** Haswell does not change the default values.
09	0	**Clock Stop OK (CSOK):** Clock stopping in D3 is not OK
08	0	**Error (ERR):** No error will ever be reported.
07:06	0	Reserved
05:04	11	**Actual Power State (APS):** Indicates the current power state of the node.
03:02	0	Reserved
01:00	11	**Requested Power State (CPS):** Reflects value written with SET_PS verb.

707h/F07h: SET/GET_PWC - Set/Get Pin Widget Control

Bits	Reset	Description
07	0	Reserved
06	1	**Out Enable (OE):** When set, the audio is enabled
05:02	0	Reserved
01:00	00	**Encoded Packet Type (EPT):**

708h/F08h: SET/GET_UE - Set/Get Unsolicited Enable

Bits	Description
07	**Unsolicited Enable (UE):** When set, unsolicited responses are allowed
06	Reserved
05:00	**Tag (TAG):**

F09h: GET_PS - Get Pin Sense

Bits	Reset	Description
31	0	**Presence Detect (PD):** When set presence is detected on this pin.
30	0	**ELD Value (ELDV):**
29	0	**Inactive (INA):**
28:00	28:00	Reserved

71Ch: SET_CD0 - Set Configuration Default Byte 0

Bits	Description
07:04	Default Association (DA):
03:00	Sequence (SEQ):

71Dh: SET_CD1 - Set Configuration Default Byte 1

Bits	Description
07:04	Color (COL):
03:00	Miscellaneous (MISC):

71Eh: SET_CD2 - Set Configuration Default Byte 2

Bits	Description
07:04	Default Device (DD):
03:00	Connection Type (CT):

71Fh: SET_CD3 - Set Configuration Default Byte 3

<table>
<tr><th>Bits</th><th>Description</th></tr>
<tr><td>07:06</td><td>Port Connectivity (PC): External connectivity of the pin complex.
00 = Connected to jack
01 = No physical connection
10 = Fixed function device (integrated speaker, mic, etc.)
11 = Both a jack and internal connection</td></tr>
<tr><td>05:00</td><td>Location (LOC):
<table>
<tr><td></td><td colspan="11">Bits 3:0</td></tr>
<tr><td>Bits 5:4</td><td>0h: N/A</td><td>1h: Rear</td><td>2h: Front</td><td>3h: Left</td><td>4h: Right</td><td>5h: Top</td><td>6h: Bottom</td><td>7h: Special</td><td>8h: Special</td><td>9h: Special</td><td>Ah-Fh Reserved</td></tr>
<tr><td>00: External</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
<tr><td>01: Internal</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td></tr>
<tr><td>10: Separate Chassis</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td><td></td><td></td></tr>
<tr><td>11: Other</td><td>Y</td><td></td><td></td><td></td><td></td><td></td><td>Y</td><td>Y</td><td>Y</td><td></td><td></td></tr>
</table></td></tr>
</table>

F1Ch: GET_CD - Get Configuration Default

Bits	Description
31:30	**Port Connectivity (PC):** See SET_CD3.PC
29:24	**Location (L):** See SET_CD3.L
23:20	**Default Device (DD):** See Set_CD2.DD
19:16	**Connection Type (CT):** See Set_CD2.CT
15:12	**Color (COL):** See SET_CD1.COL
11:08	**Miscellaneous (MISC):** See SET_CD1.MISC
07:04	**Default Association (DA):** See SET_CD0.DA
03:00	**Sequence (SEQ):** See SET_CD0.SEQ

F2Eh: HDMI/DP Info Size

Bits	Reset	Description
31:08	0	Reset
07:00	Varies	**Size (SZ):** Indexes 0 - 3 return 1Eh, index 1000 returns 53h, others reserved.

F2Fh: Get ELD Data

Parameter	Symbol	Register Name
07:0h	PARAM_INDX	ELD DATA Index

Parameter nn: ELD Data

Bits	Reset	Description
31:00	0	ELD Data

730h/F30h: SET/GET_HII - Set/Get HDMI Info Index

Bits	Reset	Description
07:05	000	**Infoframe Packet Index (IPI):** Value / Name / Value / Name 000 / Audio / 011 / GP3 001 / GP / 100 / GP4 010 / GP2 / Others / Reserved
04:00	00h	**Byte Offset Index Pointer (BOI):**

731h/F31h: SET/GET_HID - Set/Get HDMI Info Data

Bits	Reset	Description
07:00	00h	**Data (DATA):** Data at current index pointed to from SET_HII verb.

732h/F32h: SET/GET_HITC - Set/Get HDMI Info Transmit Control

Bits	Reset	Description
07:06	00	**InfoFrame Control Current Indexed Frame (IFCCIF):** • 00 = Disable Transmit • 01 = Reserved • 10 = Transmit Once • 11 = Best Effort
05:00	*0*	*Reserved*

733h SET_PC - Set Protection Control

Bits	Description
07:03	**Unsolicited Response Sub Tag (URST):** Subtag to use for unsolicited responsed.
02	Reserved

734h/F34h: SET/GET_CCM - Get/Set Converter Channel Map

Bits	Reset	Description
07:04	0h	Converter Channel (CC):
03:00	0h	Slot (SN):

735h: SET_DS - Set Device Select

Bits	Reset	Description
07:06	*0*	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present)

F35h: GET_DS - Get Device Select

Bits	Description
31:12	**Reserved:** Set to 0
11:06	**SinK Device ID:** Sink Device ID in the multi stream topology of the DP hierarchy.
05:00	**Device (D):**Device Entry index currently set

F36h: GET_DDLE - Get Display Device List Entry

Bits	Bits	Description
31:12	0	*Reserved*
11	0	**Reserved?**
10	0	**IA of Entry 2**
09	0	**ELDV of Entry 2**
08	0	**PD of Entry 2**
07	0	**Reserved?**
06	0	**IA of Entry 1**
05	0	**ELDV of Entry 1**
04	0	**PD of Entry 1**
03	0	**Reserved?**
02	0	**IA of Entry 0**
01	0	**ELDV of Entry 0**
00	0	**PD of Entry 0**

73Ch/F3Ch: SET/GET_DPID - Set/Get DisplayPort Stream ID

Bits	Reset	Description
07:03	00h	**Tag (TAG):** Represents the SSID that will go in the lower 5 bits of the SSID
02:00	000	**Index (IDX):** Pointer to program multiple SSID

Node ID 08h: Intel Vendor Widget Verbs

Set Verb	Get Verb	Symbol	Verb Name
-	F00h	GET_PARAM	Get Parameters
-	F80h	GET_HDPS	Get HDMI/DP Status
781h	F81h	SET_HVV / GET_HVV	Set/Get HDMI Vendor Verb
			delete?
-	F83h	GET_CWC	Get Captured Wall Clock
	F84h	GET_CGTC	Get Captured GTC Value
	F85h	GET_GOF	Get GTC Offset Value
785h	-	SET_GOF0	Set GTC Offset Value Byte 0
786h	-	SET_GOF1	Set GTC Offset Value Byte 1
787h	-	SET_GOF2	Set GTC Offset Value Byte 2
788h	-	SET_GOF3	Set GTC Offset Value Byte 3
789h	F89h	SET_GDI / GET_GDI	Set/Get GTC Offset Device Index

F00h: Get Parameters

Parameter	Symbol	Register Name
09h	PARAM_AWC	Audio Widget Capabilities

Parameter 09h: AWC - Audio Widget Capabilities

Bits	Reset	Description
31:24	0	Reserved
23:20	Fh	**Widget Type (TYPE):** Indicates this is a vendor defined widget
19:00	0	Reserved

781h/F81h: GET/SET_VV - Get/Set HDMI Vendor Verb

Bits	Bits	Description
07:03	0	Reserved
02	0	**Enable Widi:** When set, Widi Widget Node ID 9 is enabled.
01	0	**Enable DP1.2 Features (EDP12):** When set, DP1.2 features are enabled.
00	0	**Enable 3rd Pin and Converter Widget (E3P):** When set, the third pin and converter widget is enabled and can respond to HD Audio Verbs. When cleared, the third pin and converter widget is disabled and cannot respond to HD Audio verbs.

F83h: GET_CGTC - Get Captured GTC Value

Bits	Bits	Description
07	0	**GTC Value:** 32-bit GTC value captured on the SET_GTCT verb

F84h: GET_CWC - Get Captured Wall Clock Value

Bits	Bits	Description
31:00	0	**Wall Clock Value:** 32-bit wall clock value captured on the SET_GTCT verb

F85h: GET GOF - Get GTC Offset Value

Bits	Reset	Description
31:00	0h	**Value (VAL):** Reports the GTC Offset Value.

785h: SET GOF0 - Set GTC Offset Value Byte 0

Bits	Description
07:00	GTC Offset Value Bits [7:0]

786h: SET GOF1 - Set GTC Offset Value Byte 1

Bits	Description
07:00	GTC Offset Value Bits [15:8]

787h: SET GOF2 - Set GTC Offset Value Byte 2

Bits	Description
07:00	GTC Offset Value Bits [23:16]

788h: SET GOF3 - GTC Offset Value Byte 3

Bits	Description
07:00	GTC Offset Value Bits [31:24]

789h/F89h: SET/GET_GDI - Set/Get GTC Device Index

Bits	Reset	Description
07:06	*0*	*Reserved*
05:00	00h	**Device (D):** 000001, 000010 (based upon number of devices present). Device Index should be updated only when the Offset Value has to programmed. Behavior is unpredictable if the Device Index is updated without updating the Corresponding Offset Value.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 13: Memory-mapped Input/Output (MMIO)

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform
(Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Slice Registers and Die Recovery..... 1

Slice Registers and Die Recovery

When slice 0 is disabled (for example, GT3 fused to GT2 with a slice 0 fault), any read to a slice-located MMIO register must be directed to slice 1, otherwise data of '0' will be returned. This applies to SRM cycles from any command streamer.

MMIO Range Start	MMIO Range End	Unit Description
00005500	00005FFF	WMBE
00007000	00007FFF	SVL
00009400	000097FF	CP unit reg. file - Copy in Slice Common (in all slices)
0000B000	0000B0FF	L3 unique status registers for each slice (unicast per GT).
0000B100	0000B3FF	L3 bank config space (multicast copy per bank and slice)
0000E000	0000E0FF	DM
0000E100	0000E1FF	SC
0000E200	0000E3FF	GWL (inst. 0)
0000E200	0000E3FF	GWL (inst. 1)
0000E200	0000E3FF	GWL (inst. 2)
0000E400	0000E7FF	TDL

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 14: Observability Performance Counters

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Trace 1

Performance Visibility 1

Motivation For Hardware-Assisted Performance Visibility 1

HW Support 1

Performance Counter Registers 1

Performance Counter Reporting 4

MI_REPORT_PERF_COUNT 5

Aggregating Counters 5

Flexible EU Event Counters 8

Trace

This section contains the following contents:

- Performance Visibility

Performance Visibility

Motivation For Hardware-Assisted Performance Visibility

As the focus on GFX performance and programmability has increased over time, the need for hardware (HW) support to rapidly identify bottlenecks in HW and efficiently tune the work sent to same has become correspondingly important. This part of the PRM describes the HW support for Performance Visibility.

HW Support

This section contains various reporting counters and registers for hardware support for Performance Visibility.

Performance Counter Registers

The following Performance Statistics registers must be part of the power context:

OAPERF_A0 - Aggregate Perf Counter A0

OAPERF_A0_UPPER - Aggregate Perf Counter A0 Upper DWord

OAPERF_A1 - Aggregate Perf Counter A1

OAPERF_A1_UPPER - Aggregate Perf Counter A1 Upper DWord

OAPERF_A2 - Aggregate Perf Counter 2

OAPERF_A2_UPPER - Aggregate Perf Counter A2 Upper DWord

OAPERF_A3 - Aggregate Perf Counter A3

OAPERF_A3_UPPER - Aggregate Perf Counter A3 Upper DWord

OAPERF_A4 - Aggregate Perf Counter A4

OAPERF_A4_UPPER - Aggregate Perf Counter A4 Upper DWord

OAPERF_A5 - Aggregate Perf Counter A5

OAPERF_A5_UPPER - Aggregate Perf Counter A5 Upper DWord

OAPERF_A6 - Aggregate Perf Counter A6

OAPERF_A6_UPPER - Aggregate Perf Counter A6 Upper DWord

OAPERF_A7 - Aggregate Perf Counter A7

OAPERF_A8 - Aggregate Perf Counter A8

OAPERF_A8_UPPER - Aggregate Perf Counter A8 Upper DWord

OAPERF_A9 - Aggregate Perf Counter A9

OAPERF_A9_UPPER - Aggregate Perf Counter A9 Upper DWord

OAPERF_A10 - Aggregate Perf Counter A10

OAPERF_A10_UPPER - Aggregate Perf Counter A10 Upper DWord

OAPERF_A11 - Aggregate Perf Counter A11

OAPERF_A11_UPPER - Aggregate Perf Counter A11 Upper DWord

OAPERF_A12 - Aggregate Perf Counter A12

OAPERF_A12_UPPER - Aggregate Perf Counter A12 Upper DWord

OAPERF_A13 - Aggregate Perf Counter A13

OAPERF_A13_UPPER - Aggregate Perf Counter A13 Upper DWord

OAPERF_A14 - Aggregate Perf Counter A14

OAPERF_A14_UPPER - Aggregate Perf Counter A14 Upper DWord

OAPERF_A15 - Aggregate Perf Counter A15

OAPERF_A15_UPPER - Aggregate Perf Counter A15 Upper DWord

OAPERF_A16 - Aggregate Perf Counter A16

OAPERF_A16_UPPER - Aggregate Perf Counter A16 Upper DWord

OAPERF_A17 - Aggregate Perf Counter A17

OAPERF_A17_UPPER - Aggregate Perf Counter A17 Upper DWord

OAPERF_A18 - Aggregate Perf Counter A18

OAPERF_A18_UPPER - Aggregate Perf Counter A18 Upper DWord

OAPERF_A19 - Aggregate Perf Counter A19

OAPERF_A19_UPPER - Aggregate Perf Counter A19 Upper DWord

OAPERF_A20 - Aggregate Perf Counter A20

OAPERF_A20_UPPER - Aggregate Perf Counter A20 Upper DWord

OAPERF_A21 - Aggregate Perf Counter A21

OAPERF_A21_UPPER - Aggregate Perf Counter A21 Upper DWord

OAPERF_A22 - Aggregate Perf Counter A22

OAPERF_A22_UPPER - Aggregate Perf Counter A22 Upper DWord

OAPERF_A23 - Aggregate Perf Counter A23

OAPERF_A23_UPPER - Aggregate Perf Counter A23 Upper DWord

OAPERF_A24 - Aggregate Perf Counter A24

OAPERF_A24_UPPER - Aggregate Perf Counter A24 Upper DWord

OAPERF_A25 - Aggregate Perf Counter A25

OAPERF_A25_UPPER - Aggregate Perf Counter A25 Upper DWord

OAPERF_A26 - Aggregate Perf Counter A26

OAPERF_A26_UPPER - Aggregate Perf Counter A26 Upper DWord

OAPERF_A27 - Aggregate Perf Counter A27

OAPERF_A27_UPPER - Aggregate Perf Counter A27 Upper DWord

OAPERF_A28 - Aggregate Perf Counter A28

OAPERF_A28_UPPER - Aggregate Perf Counter A28 Upper DWord

OAPERF_A29 - Aggregate Perf Counter A29

OAPERF_A29_UPPER - Aggregate Perf Counter A29 Upper DWord

OAPERF_A30 - Aggregate Perf Counter A30

OAPERF_A30_UPPER - Aggregate Perf Counter A30 Upper DWord

OAPERF_A31 - Aggregate_Perf_Counter_A31

OAPERF_A31_UPPER - Aggregate Perf Counter A31 Upper DWord

OAPERF_A32 - Aggregate_Perf_Counter_A32

OAPERF_A33 - Aggregate_Perf_Counter_A33

OAPERF_A34 - Aggregate_Perf_Counter_A34

OAPERF_A35 - Aggregate_Perf_Counter_A35

OAPERF_B0 - Boolean_Counter_B0

OAPERF_B1 - Boolean_Counter_B1

OAPERF_B2 - Boolean_Counter_B2

OAPERF_B3 - Boolean_Counter_B3

OAPERF_B4 - Boolean_Counter_B4

OAPERF_B5 - Boolean_Counter_B5

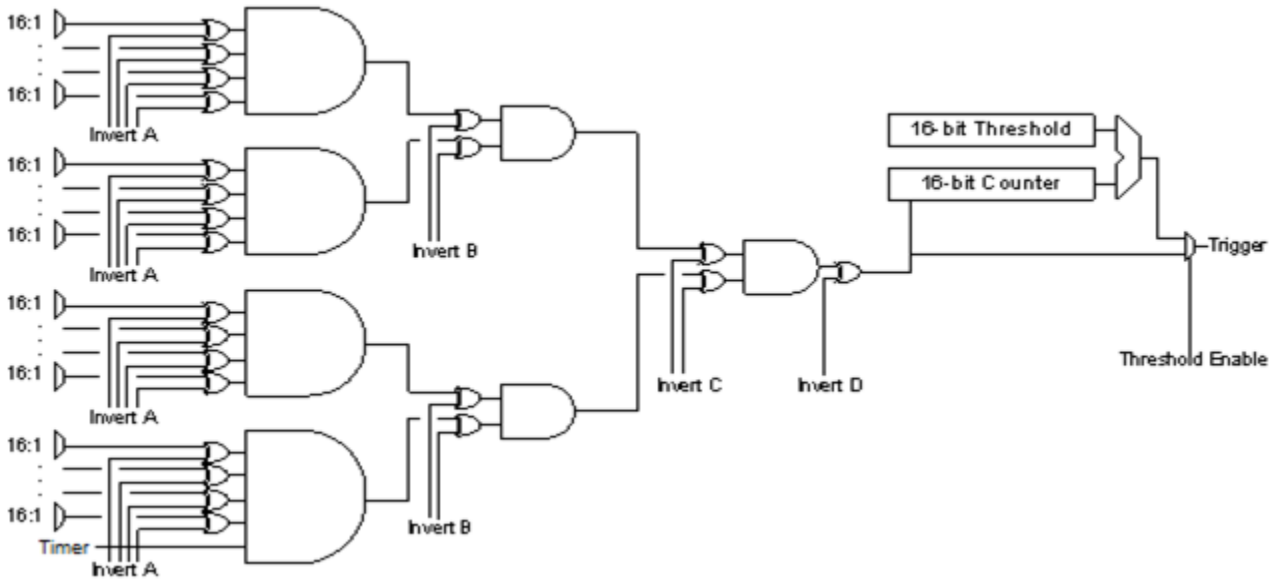
OAPERF_B6 - Boolean_Counter_B6

OAPERF_B7 - Boolean_Counter_B7

Performance Counter Reporting

When either the MI_REPORT_PERF_COUNT command is received or the internal report trigger logic fires, a snapshot of the performance counter values is written to memory. The format used by HW for such reports is selected using the Counter Select field within the OACONTROL register. The organization and number of report formats vary per project and are detailed in the following section. In the following layouts, the RPT_ID is always stored in the lowest addressed DWORD.

OA contains logic to control when performance counter values are reported to memory. This functionality is controlled using the OA report trigger and OA start trigger registers. More detailed register descriptions are included in the Hardware Programming interface. The block diagram below illustrates the logic these registers control.



Note that counters which are 40 bits wide are split in the report format into low DWORD and high byte chunks for simplicity of HW implementation as well as SW-friendly alignment of report data. The performance counter read logically done before writing out report data for these 40-bit counters is guaranteed to be an atomic operation, the counter data is simply swizzled as it is being packed into the report.

MI_REPORT_PERF_COUNT

MI_REPORT_PERF_Count

Aggregating Counters

The table below described the desired high-level functionality from each of the aggregating counters.

Note that there is no counter of 2x2s sent to pixel shader, this is based on the assumption that the pixel shader invocation pipeline statistics counter increments for partially lit 2x2s as well and hence does not require a duplicate performance counter.

Counter #	Event	Description
A0	Render Engine Busy	Render engine is not idle. GPU Busy aggregate counter doesn't increment under the following conditions: 1. Context Switch in Progress. 2. GPU stalled on executing MI_WAIT_FOR_EVENT. 3. GPU stalled on execution MI_SEMAPHORE_MBOX. 4. RCS idle but other parts of GPU active (e.g. only media engines active)
A1	# of Vertex Shader Threads Dispatched	Count of VS threads dispatched to EUs
A2	# of Hull Shader Threads Dispatched	Count of HS threads dispatched to EUs
A3	# of Domain Shader Threads Dispatched	Count of DS threads dispatched to EUs
A4	# of GPGPU Threads Dispatched	Count of GPGPU threads dispatched to EUs
A5	# of Geometry Shader Threads Dispatched	Count of GS threads dispatched to EUs
A6	# of Pixel Shader Threads Dispatched	Count of PS threads dispatched to EUs
A7	Aggregating EU counter 0	User-defined (details in Flexible EU Event Counters section)
A8	Aggregating EU counter 1	User-defined (details in Flexible EU Event Counters section)
A9	Aggregating EU counter 2	User-defined (details in Flexible EU Event Counters section)
A10	Aggregating EU counter 3	User-defined (details in Flexible EU Event Counters section)
A11	Aggregating EU counter 4	User-defined (details in Flexible EU Event Counters section)

Counter #	Event	Description
A12	Aggregating EU counter 5	User-defined (details in Flexible EU Event Counters section)
A13	Aggregating EU counter 6	User-defined (details in Flexible EU Event Counters section)
A14	Aggregating EU counter 7	User-defined (details in Flexible EU Event Counters section)
A15	Aggregating EU counter 8	User-defined (details in Flexible EU Event Counters section)
A16	Aggregating EU counter 9	User-defined (details in Flexible EU Event Counters section)
A17	Aggregating EU counter 10	User-defined (details in Flexible EU Event Counters section)
A18	Aggregating EU counter 11	User-defined (details in Flexible EU Event Counters section)
A19	Aggregating EU counter 12	User-defined (details in Flexible EU Event Counters section)
A20	Aggregating EU counter 13	User-defined (details in Flexible EU Event Counters section)
A21	2x2s Rasterized	Count of the number of samples of 2x2 pixel blocks generated from the input geometry before any pixel-level tests have been applied. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.)
A22	2x2s Failing Fast pre-PS Tests	Count of the number of samples failing fast "early" (i.e. before pixel shader execution) tests (counted at 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.)
A23	2x2s Failing Slow pre-PS Tests	Count of the number of samples of failing slow "early" (i.e. before pixel shader execution) tests (counted at 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) If a 2x2 sample partially fails the Z/STC test (i.e some pixels fail and some pixels pass), the OA slow fail counter value will be incorrect.
A24	2x2s Killed in PS	Number of samples entirely killed in the pixel shader as a result of explicit instructions in the kernel (counted in 2x2 granularity). (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Behavior of this counter changes when MSAA is enabled based on PS dispatch mode (per-sample versus per-pixel). This leads to discrepancies in how A24/A25 increment versus how A21-A23 and A26/A27 increment when both MSAA and per-pixel PS dispatch are enabled. Counter may be inaccurate when pixel shader outputs output mask (e.g. DX11 oMask declaration)

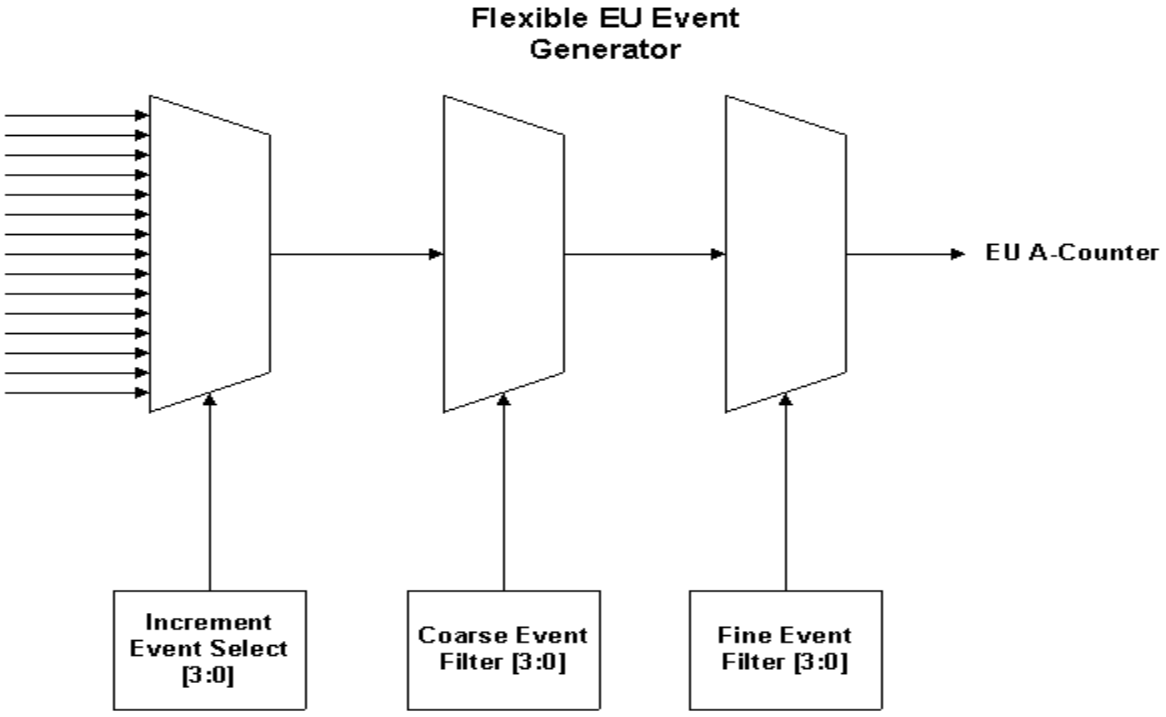
Counter #	Event	Description
A25	2x2s Failing post-PS Tests	Number of samples that entirely fail "late" tests (i.e. tests that can only be performed after pixel shader execution). Counted at 2x2 granularity. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.) Counter may be inaccurate when pixel shader is allowed to modify output mask (e.g. DX11 oMask declaration) Behavior of this counter changes when MSAA is enabled based on PS dispatch mode (per-sample versus per-pixel). This leads to discrepancies in how A24/A25 increment versus how A21-A23 and A26/A27 increment when both MSAA and per-pixel PS dispatch are enabled.
A26	2x2s Written To Render Target	Number of samples that are written to render target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.)
A27	Blended 2x2s Written to Render Target	Number of samples of blendable that are written to render target.(counted at 2x2 granularity). MRT case will report multiple writes per 2x2 processed by the pixel shader. (Please note that 2x2s may be in terms of pixels or in terms of samples depending on project but are consistent between A21-A27.)
A28	2x2s Requested from Sampler	Aggregated total 2x2 texel blocks requested from all EUs to all instances of sampler logic.
A29	Sampler L1 Misses	Aggregated misses from all sampler L1 caches. Please note that the number of L1 accesses varies with requested filtering mode and in other implementation specific ways. Hence it is not possible in general to draw a direct relationship between A28 and A29. However, a high number of sampler L1 misses relative to texel 2x2s requested frequently degrades sampler performance.
A30	SLM Reads	Total read requests from an EU to SLM (including reads generated by atomic operations).
A31	SLM Writes	Total write requests from an EU to SLM (including writes generated by atomic operations).
A32	Other Shader Memory Accesses	Reserved, can generate per HDC version by looking at (hdc_cput == 1) && (hdc_dest[2:0] == 0b000 || hdc_dest[2:0] == 0b010).
A32	Other Shader Memory Accesses	Aggregated total requests from all EUs to memory surfaces other than render target or texture surfaces (e.g. shader constants).
A34	Atomic Accesses	Aggregated total atomic accesses from all EUs. This counter increments on atomic accesses to both SLM and URB.
A35	Barrier Messages	Aggregated total completed barriers (one per barrier).
A35	Barrier Messages	Aggregated total kernel barrier messages from all Eus (one per thread in barrier).

Flexible EU Event Counters

Since EU performance events are most interesting in many cases when aggregated across all EUs and many interesting EU performance events are limited to certain APIs (e.g. hull shader kernel stats only applicable when running a DX11+ workload), CHV, BSW adds some additional flexibility to the aggregated counters coming from the EU array.

The following block diagram shows the high-level flow that generates each flexible EU event.

Note that no support is provided for differences between flexible EU event programming between EUs because the resulting output from each EU is eventually merged into a single OA counter anyway.



Supported Increment Events

Increment Event	Encoding	Notes
EU FPU0 Pipeline Active	0b0000	Signal that is high on every EU clock where the EU FPU0 pipeline is actively executing a Gen ISA instruction.
		Please note that FPU0 in this EU is the closest match to previous Gen EU's FPU pipe.
EU FPU1 Pipeline Active	0b0001	Signal that is high on every EU clock where the EU FPU1 pipeline is actively executing a Gen ISA instruction.
		Please note that FPU1 in this EU is the closest match to previous Gen EU's EM pipe.
EU SEND Pipeline Active	0b0010	Signal that is high on every EU clock where the EU send pipeline is actively executing a Gen ISA instruction. Only fine event filters 0b0000,0b0101, 0b0110, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
EU FPU0 & FPU1 Pipelines Concurrently Active	0b0011	Signal that is high on every EU clock where the EU FPU0 and FPU1 pipelines are both actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Some EU Pipeline Active	0b0100	Signal that is high on every EU clock where at least one EU pipeline is actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000,0b0101, 0b0110, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
At Least 1 Thread Loaded But No EU Pipeline Active	0b0101	Signal that is high on every EU clock where at least one thread is loaded but no EU pipeline is actively executing a Gen ISA instruction. Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.
Threads loaded integrator == max threads for current HW SKU	0b1000	Implies an accumulator which increases every EU clock by the number of loaded threads, signal pulses high for one clock when the accumulator exceeds a multiple of the number of thread slots (e.g. for a 8-thread EU, signal pulses high every clock where the increment causes a 3-bit accumulator to overflow). Only coarse event filters 0b0000, 0b0111, and 0b1000 are supported with this increment event. Only fine event filters 0b0000, 0b0111, 0b1000, 0b1001, and 0b1010 are supported with this increment event.

Supported Coarse Event Filters

Coarse Event Filter	Encoding	Notes
No mask	0b0000	Never masks increment event.
Currently executing thread came from VS	0b0001	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of VS.
Currently executing thread came from HS	0b0010	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of HS.
Currently executing thread came from DS	0b0011	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of DS.
Currently executing thread came from GS	0b0100	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of GS.
Currently executing thread came from PS	0b0101	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of PS.
Currently executing thread came from TS	0b0110	Masks increment event unless the FFID which dispatched the currently executing thread equals FFID of TS.
Row = 0	0b0111	Masks increment event unless the row ID for this EU is 0 (control register is in TDL so only have to check within quarter-slice).
Row = 1	0b1000	Masks increment event unless the row ID for this EU is 1 (control register is in TDL so only have to check within quarter-slice).

Fine Event Filters

Fine Event Filter	Encoding	Notes
None	0b0000	Never mask increment event.
Cycles where hybrid instructions are being executed	0b0001	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are hybrid instructions.
		Filter behaves unreliably when shader ISA uses 64-bit immediate values.
Cycles where ternary instructions are being executed	0b0010	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are ternary instructions.
Cycles where binary instructions are being executed	0b0011	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are binary instructions.
Cycles where mov instructions are being executed	0b0100	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are mov instructions.
Cycles where sends start being executed	0b0101	Masks increment event unless the instruction(s) being executed on the pipeline(s) selected by the increment event are send start of dispatch. Note that if this fine event filter is used in combination with increment events not related to the EU send pipeline (e.g. FPU0 active), the associated flexible event counter will increment in an implementation-specific manner.
EU# = 0b00	0b0111	Masks increment event unless the EU number for this EU is 0b00.
EU# = 0b01	0b1000	Masks increment event unless the EU number for this EU is 0b01.
EU# = 0b10	0b1001	Masks increment event unless the EU number for this EU is 0b10.
EU# = 0b11	0b1010	Masks increment event unless the EU number for this EU is 0b11.

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 15: Graphics PCI Registers

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

GFX PCI Registers..... 1

GFX PCI Registers

Address Space	Address	Symbol	Name
PCI: 0/2/0	00000h	DID	Device ID and Vendor ID
PCI: 0/2/0	00004h	PCICMD_STS	PCI Command and Status
PCI: 0/2/0	00008h	RID_CC	Revision ID and Class Code
PCI: 0/2/0	0000Ch	HDR_CC	Header Type
PCI: 0/2/0	0000Ch	GTTMMADR_LSB	Base Graphics Translation Table and Memory Mapped Range
PCI: 0/2/0	00014h	GTTMMADR_MSB	Base Graphics Translation Table Modification and Memory Mapped Range
PCI: 0/2/0	00018h	GMADR_LSB	Graphics Aperture Location Masks
PCI: 0/2/0	0001Ch	GMADR_MSB	Graphics Aperture Location
PCI: 0/2/0	00020h	IOBAR	I/O Base Address
PCI: 0/2/0	0002Ch	SSID_SID	Subsystem ID
PCI: 0/2/0	00034h	CAPPOINT	Pointer to linked list of Capabilities for device
PCI: 0/2/0	0003Ch	INTRLINE	Interrupt Line Routing
PCI: 0/2/0	00050h	GGC	GMCH Graphics Control Register
PCI: 0/2/0	0005Ch	BDSM	Base address of Data Stolen DRAM Memory
PCI: 0/2/0	00060h	MSAC	Size of the Graphics Memory Aperture
PCI: 0/2/0	00070h	BGSM	Base of the GTT table in Gfx Stolen Memory
PCI: 0/2/0	00090h	MSI_CAPID_MC	Message Signaled Interrupts Capability ID and Control Register
PCI: 0/2/0	00094h	MA	Message Address
PCI: 0/2/0	00098h	MD	Message Data
PCI: 0/2/0	000A4h	AFLC	Advanced Features, Length and Capabilities
PCI: 0/2/0	000A8h	AFCTLSTS	Advanced Features Control and Status
PCI: 0/2/0	000B0h	VCID	Vendor Capability ID
PCI: 0/2/0	000B4h	VC	Vendor Capabilities
PCI: 0/2/0	000C4h	FD	Functional Disable
PCI: 0/2/0	000D0h	PMCAPID	Power Management Capabilities and ID
PCI: 0/2/0	000D4h	PMCS	Power Management Control Status
PCI: 0/2/0	000E0h	SWSMISCI	Software System Management Interrupt and System Control Interrupt Event Source and Trigger
PCI: 0/2/0	000E4h	ASLE	System Display Event Register
PCI: 0/2/0	000FCh	ASLS	Advanced Configuration and Power Interface Source Language Storage

Intel® Open Source HD Graphics Programmers' Reference Manual (PRM)

Volume 16: Workarounds

For the 2014-2015 Intel Atom™ Processors, Celeron™ Processors and Pentium™ Processors based on the "Cherry Trail/Braswell" Platform (Cherryview/Braswell graphics)

October 2015, Revision 1.1

Creative Commons License

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

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

Notices and Disclaimers

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

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

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

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

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

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

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

Copyright © 2015, Intel Corporation. All rights reserved.

Table of Contents

Workarounds **1**

Workarounds

Functional Area	Component	Workaround Name	Workaround Description
3D	Data Port Messages		Render Target Write messages require a header when Pixel Shader Computed Depth is enabled.
3D	SURFACE_STATE		The R32_FLOAT, R32G32_FLOAT and R8G8_UNORM surface formats are not handled correctly for the Alpha channel for cases where the sample is off-map or out of bounds. The correct behavior should be to force these Alpha values to 1.0, but actual result is for these formats is a blending of the default color with 1.0.
3D	3D Sampler Message Types		When a surface of type surfaceType_Null is accessed by resinfo, the MIPCount returned is undefined instead of 0.
3D	Message Header		Offu/offv are calculated in normalized space and hence subject to small truncation error.
3D	3D Sampler Messages and Message Types		Offu/offv are calculated in normalized space and hence subject to a small truncation error.
3D	Vertex Fetch Functions		If a 32 bit uscaled or sscaled format is used, then a float format needs to be used so VF will keep the data as is and the kernel needs to convert the format to 32 bit float.
3D		WaDisableAsyncFlipPerfMode	Async flips hang on MI_WAIT_FOR_EVENT following it. WaDuplicateMiDisplayFlip does not seem to help. To workaround this we have enabled MI_SYNC_FLIP always.

3D	Patch Header DW0-7		The Tessellation stage will incorrectly add domain points along patch edges under the following conditions, which may result in conformance failures and/or cracking artifacts:
			QUAD domain
			INTEGER partitioning
			All three Tess Factors in a given U or V direction (for example,, V direction: UEQ0, InsideV, UEQ1) are all exactly 1.0
			All three Tess Factors in the other direction are > 1.0 and all round up to the same integer value (e.g, U direction: VEQ0 = 3.1, InsideU = 3.7, VEQ1 = 3.4)
			The suggested workaround (to be implemented as part of the post-amble to the HS shader in the HS kernel) is:
			if (
			(TF[UEQ0] > 1.0) ||
			(TF[VEQ0] > 1.0) ||
			(TF[UEQ1] > 1.0) ||
			(TF[VEQ1] > 1.0) ||
			(TF[INSIDE_U] > 1.0) ||
			(TF[INSIDE_V] > 1.0))
			{
			TF[INSIDE_U] = (TF[INSIDE_U] == 1.0) ? 2.0 : TF[INSIDE_U];
			TF[INSIDE_V] = (TF[INSIDE_V] == 1.0) ? 2.0 : TF[INSIDE_V];

<table>
<tr>
<td>3D</td>
<td></td>
<td>WaPreventHSTessLevelsInterference</td>
<td>Inner tessellation levels can interfere with outer tess levels. The proposed SW workaround is to have the HS compiler, upon seeing INTEGER and QUAD, generate instructions to perform the following logic after or when moving the TFs into the Patch Header:
if (
(TF[UEQ0] > 1.0) ||
(TF[VEQ0] > 1.0) ||
(TF[UEQ1] > 1.0) ||
(TF[VEQ1] > 1.0) ||
(TF[INSIDE_U > INSIDE_U] > 1.0) ||
(TF[INSIDE_V > INSIDE_V] > 1.0))
{
TF[INSIDE_U|INSIDE_U] = (TF[INSIDE_U|INSIDE_U] == 1.0)
? 2.0: TF[INSIDE_U|INSIDE_U];
TF[INSIDE_V|INSIDE_V] = (TF[INSIDE_V|INSIDE_V] == 1.0)
? 2.0: TF[INSIDE_V|INSIDE_V];
}
Note that the non-inside TFs are never modified, so this won't impact the HW patch cull testing for 0= or NaN. (This serves as a warning to watch out for NaN comparisons).</td>
</tr>
<tr>
<td>3D</td>
<td>GPGPU Indirect Thread Dispatch</td>
<td></td>
<td>The indirect registers are not supposed to be set to 0, but sometimes the kernel computing the value wants no work done and sets them to 0. This does not work correctly, so a workaround in the command stream is needed. The following workaround sets the predicate if IndirectArgs.X=0 or IndirectArgs.Y=0 or IndirectArgs.Z=0
<table>
<tr><th>Command</th><th>Operation</th></tr>
<tr><td>MI_LOAD_REGISTER_IMM</td><td>MI_PREDICATE_RESULT = 0</td></tr>
<tr><td>MI_LOAD_REGISTER_IMM</td><td>MI_PREDICATE_SRC1 = 0</td></tr>
<tr><td>MI_LOAD_REGISTER_MEM</td><td>MI_PREDICATE_SRC0 = [IndirectArgs].X</td></tr>
<tr><td>MI_PREDICATE</td><td>LOAD, OR, SOURCES_EQUAL</td></tr>
<tr><td>MI_LOAD_REGISTER_MEM</td><td>MI_PREDICATE_SRC0 = [IndirectArgs].Y</td></tr>
</table>
</td>
</tr>
</table>

			MI_PREDICATE	LOAD, OR, SOURCES_EQUAL
			MI_LOAD_REGISTER_MEM	MI_PREDICATE_SRC0 = [IndirectArgs].Z
			MI_PREDICATE	LOADINV, OR, SOURCES_EQUAL
			WALKER	Predicate Enable = 1
3D	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation		The Typed Surface Read message returns 0 in all channels for out-of-bounds accesses.	
3D	Shared Local Memory (SLM)		SLM accesses are incorrectly treated as out-of-bounds when the General State Buffer Size in STATE_BASE_ADDRESS is set to zero. The workaround is to set the General State Buffer Size to a non-zero value.	
3D	Media State and Primitive Commands		The MEDIA_STATE_FLUSH command is updated to optionally specify all the resources required for the next thread group via an interface descriptor – if the resources are not available the group cannot start.	
3D	Shared Local Memory (SLM)		When SLM memory is disabled in the L3 configuration, SLM accesses are correctly treated as out-of-bounds but the out-of-bounds data returned may not be zero.	
3D		WaDisableVFUnitClockGating	VF drawing too many verts on preemption restore.	
3D	Data Port Messages		For Atomic Counter OPS other than INC, DEC, and PREDEC, the header is forbidden and not optional as indicated in the table.	
3D		WaDividePSInvocationCountBy4	Invocation counter is 4 times actual.	
			WA: SW to divide HW reported PS Invocations value by 4.	
3D		WaVSRefCountFullforceMissDisable	Hang; For GS\VS workloads. WA: Set FF_MODE - Thread Mode Register, 0x020A0: DS Reference Count Full Force Miss Enable, bit 19 = 0, VS Reference Count Full Force Miss Enable, bit 15 = 0. HW issue that requires VS_REFERENCE_COUNT_FULL_FORCE_MISS_ENABLE (#15) bit of MMIO 0x20a0 to be clear. Since this bit is always clear by default, the driver shouldn't do anything as long as the WA is enabled and should set this bit if the WA is disabled. To be done at the boot time.	

3D		WaHizAmbiguateRequiredNonAlignedBeforeRender	Corruption. WA: Must do HiZ resolve (ambiguate) on any portion of the HiZ buffer that is not 8x4 aligned before rendering or clearing.
3D	Atomic Counter Operation Message Descriptor		For Atomic Counter OPS other than INC, DEC, or PREDEC, the message header is forbidden and not optional.
3D		WaIndirectDispatchPredicate	GFXDRV: Hang in GT2 WW03d Emulation - GPGPU _WALKER HANG. WA: Use MI_PREDICATE commands to predicate an indirect dispatch whose thread groups in any dimension = 0. GPGPU pipe3d: vfe handle counter underflow for dimx eq 0 followed by curbe workload.
3D	Universal Input Message Phases		WA: The following text needs to be maintained so that we can bring back the feature in the next opportunity. Will be used for Field 16x8 Enabled: This field enables 16x8 interlaced–block partitioning for MPEG-2. Note: Enabling Field 16x8 prevents use of sub-partitions types 8x16, 4x4, 4x8 and 8x4, RefAccess and SrcAccess must be 0 and SrcSize must be 16x16 (00). Field8x8 and Field16x8 are mutually exclusive.
3D	Notification Registers		The sub-register numbers for n0.0 and n0.2 are swapped on a write, that is, a destination of n0.0 is required to update n0.2 and n0.2 is required to update n0.0.
3D		AccWrEnNotAllowedToAcc1With16bit	Implicit write (AccWrEn) to acc1 (for example, H1, Q3) not allowed with 16-bit data type (for example, hf, w). Acc1 dependency is not set as EUTC doesn't detect it as write to 2nd half accumulator for half float Q3 and takes it as acc0 dest. It is using nib_ctrl[1] to determine which part of acc is being written. this works for float dest but for hf destination nib_ctrl[2] should be used.
3D		WaEnsureMemCoherencyBeforeLoadRegisterMem	MI_STORE_REG_MEM followed by MI_LOAD_REGISTER_MEM does not always stall before the LOAD. Requires that a sync event (like MI_ATOMIC or PIPE_CONTROL with CS stall) be issued between the events to ensure that the memory has been forced to be coherent. See B-Spec for restriction.

3D		WaFastClearZ16K	The definition of the HiZ based fast clear does not specify the ability to clear all the way in X and Y to 16K. The last HiZ in the X and Y would be only partially initialized. Workaround suggested is to clear as four (2x2) sets of 4k x 4k clears. The rectangle size in the HZ_OP packet can only be programmed up to 16k-1 pixels in x and y direction. We need one more bit in each direction to be able to clear the whole thing.
3D		WaFlushOpOnCSStall	When a pipecontrol with stall bit is set, an ISP bit is enabled and there is no post sync operation: CS is not doing a flush or dummy atomic while fulsim does and there is a mismatch. Restriction: if stall bit is set, any one of the post sync-operation should be set.
3D		WaOGLGSVertexReorderingTriStripAdjOnly	GS vtxram read pointer increments incorrectly if control topology value matches tristrip_adj* decoding value...value (6'b001100 , 6'b010101). This causes full to not be generated properly and eventually corruption of the GS internal storage structure..
3D		WaDSRefCountFullforceMissDisable	DS_REFERENCE_COUNT_FULL_FORCE_MISS_ENABLE (#19) bit of MMIO 0x20a0 needs to be clear in all steppings. Since this bit is always clear by default, the driver shouldn't do anything as long as the WA is enabled and should set this bit if the WA is disabled. To be done at the boot time.
3D		WaGuardbandSize	Cref precision loss in Z interpolator (src_cref mismatch causing HIZ memdiff). WA: Limit guardband to -16k to +16k.
3D		WaDisableRSBeforeBTPoolDisable	BT edits are dropped in the R S if the pool is disabled. WA: Disable RS before BT pool is disabled.
3D		WaForcenonzeroSBEOutputAttributeCount	WA: Clamp SF number of output attributes to min value of 1 to avoid GS running out of handles.
3D	LOD Information		WA: No range checks are performed on LOD from the Address Payload. The low 4 bits of LOD in the Address Payload are used to right shift the width, height, and depth values.
3D		WaSubtract1FromMaxNoOfThreads	Subtract 1 from Max No of Threads per PSD (Per Slice Dispatch) because the PSD RTL incorrectly adds 1 to the max thread programmed. So when programming "MaximumNumberOfThreadsPerPSD" in 3DSTATE_PS command, always subtract 1 from the value to be programmed.

3D		WaUseNonPrivRegisterForObjectLevelPreemption	Use INSTPM(20c0) MMIO regs to disable Mid-Object pre-emption for draw call.
3D		WaStateBindingTableOverfetch	HW over-fetches two cache lines of binding table indices. When using the resource streamer, SW needs to pad binding table pointer updates with an additional two cache lines.
3D	Floating-Point Support		When converting from float to int, rounding mode RZ must be used.
3D		WaVfPostSyncWrite	Set post sync op of write for PIPE_CONTROL when only VF cache invalidate set.
3D		WaCsStallBeforenonzeroInstanceCount	Issue CS Stall when going from 3DSTATE_HS zero instance count or HS disabled to 3DSTATE_HS non-zero instance count.
3D	Byte Scattered ReadWrite		The stateless model is not supported. A workaround is to use the A64 Byte Scattered message, adding the General State Base Address into each address offset in the message. However, the A64 message will not perform A32 stateless bounds checking.
3D	TypedUntyped Surface ReadWrite and TypedUntyped Atomic Operation		Tile W surfaces must be of format R8_UINT and only support SIMD8. Furthermore, only the RED channel can be enabled.
3D	3D Sampler Message Types		Message sample+killpix cannot be used with Fault & Stream mode.
3D	Stateless Model		A64 Stateless accesses are incorrectly treated as out-of-bounds when the General State Buffer Size in STATE_BASE_ADDRESS is set to zero. The workaround is to set the General State Buffer Size to a non-zero value.
3D	Message Mode: M0.2, bit 31:30		BYTE_MASK is not supported.
3D			PSD variable dispatch with per sample 4x and 8x - restriction from previous project has to be removed.
			WA: Enable only SIMD8 dispatch mode when in per-sample mode.

3D		WsVfPostSyncWrite	Hang: If the push constant is committed using PIPE_CONTROL with only VF cache invalidation then CS commits the push constant but does not send dirty bits to shaders. Thus the shaders will not see the dirty bits and if there is new push constant command followed by a stalling flush, CS will commit the second push constant and wait for the previous push constant deref. This causes the flush to hang as CS waits for "push constant done" in flush completion. WA:"Post Sync Operation" must be enabled to "Write Immediate Data" or "Write PS Depth Count" or "Write Timestamp".
3D	Media State and Primitive Commands		Every 8 MEDIA_OBJECT or MEDIA_OBJECT_WALKER commands that use indirect payload must be followed by a MEDIA_STATE_FLUSH or PIPE_CONTROL.
3D	Extended Math Function		Mixed mode operations are not supported
3D		WaForceMinMaxGSThreadCount	GS being stalled can cause the fftid to go over max threads causing undefined scratch space to be used. WA: Limit the number of handles to the number of threads, with some GS performance loss. Set min/max threads to 8 for GS. Should be handled in USC/IGC.
3D			WA: A PIPE_CONTROL with CS_STALL must be sent whenever the HS_STATE.InstanceCount changes from 0 (no instancing to > 0 (instancing).
3D		WaAdditionalMovWhenSrc1ModOnMulMach	A source modifier must not be used on src1 for the mul/mach macro operations. WA: Use extra move instead of src modifier for src1.
3D		WaRestoreFC4RegisterDW0fromDW1	GPGPU Pre-emption - Execution Mask not being saved/restored correctly (memdiff). WA: SIP routine has to correct the address while restoring. Flow control register FC4 has to be restored from DW1.
3D		WaScalarAtomic	This is a performance improvement implemented as a W/A. Improves append counter updates from 1/6 clks (L3 limit) to 16/6 clks.
3D	Media State and Primitive Commands		Two MEDIA_STATE_FLUSH commands need to be used to ensure that the flush is complete.
3D	Extended Math Function		When both srcs are NAN, FDIV produces denominator NAN as output.

3D		WaThreadSwitchAfterCall	GPGPU Pre-emption - CALL Inst ruction Hang. WA: Follow every call by a dummy non-JEU and non-send instruction with a SWITCH for both cases whether a subroutine is taken or not.
3D	MEDIA_STATE_FLUSH		A MEDIA_STATE_FLUSH with no options must be added after a GPGPU_WALKER command which doesn't use either SLM or barriers.
3D		WaNearestFilterLODClamp	DX10.1 LOD clamping VS Max LOD DX case. Workarounds: DX: If (mipfilter_nearest) MaxLOD = floor(MaxLOD) MinLOD = floor(MinLOD) OGL: If (mipfilter_nearest) lodbiad = lodbiad - 0.000001b Dx9 - (Not Required for Dx9. Max always set to to 14.0)
3D	GPGPU Context Switch Workarounds		After either a MI_SET_CONTEXT or a PIPE_CONTROL with Generic Media State Clear, there must be a MEDIA_VFE_STATE command before any pre-emptable command. The parameter of this MEDIA_VFE_STATE command can be set to default values.
3D	GPGPU Indirect Thread Dispatch		CURBE should be used for the payload when using indirect dispatch rather than indirect payload
3D			SLM accesses are incorrectly treated as out-of-bounds when the General State Buffer Size in STATE_BASE_ADDRESS is set to zero. The workaround is to set the General State Buffer Size to a non-zero value. When SLM memory is disabled in the L3 configuration, SLM accesses are correctly treated as out-of-bounds but the out-of-bounds data returned may not be zero.
3D	3D Sampler Message Types		If Surface Format is R10G10B10_SNORM_A2_UNORM and Gather4 Source Channel Select is alpha channel, the returned value may be incorrect.

3D	Programming Media Pipeline - Command Sequence		A MEDIA_STATE_FLUSH needs to be placed right before the MI_BATCH_BUFFER_END of any batch buffer that uses MEDIA_OBJECT.
3D	Depth Buffer Clear		To fast depth clear a full 16k, in the X and/or Y dimensions, the clear operation must be sectioned into rectangles smaller in X and Y than 16k pixels. For example to clear a 16k x 16k surface perform four (2x2) 4k x 4k clears with the proper address offsets.
3D	LOD Information		The LOD is in-bounds if LOD < MIPCount and if MinLOD + LOD < 15. If LOD is not in-bounds then 0 is returned for the width, height, and depth values.
3D		WaClearArfDependenciesBeforeEot	BattleForge3 hang - flag register dependency not cleared after EOT. WA: Source ARF registers before EOT.
Blitter	BLT Programming Restrictions		There are two suggested software workarounds to perform coherent overlapping BLTs: (a) The Source and Destination Base Address registers must hold the same value (without alignment restriction) and (b) the Source and Destination Pitch registers (BR11, BR13) must both be a multiple of 64 bytes. Or if (a) is not possible do overlapping source copy BLTs as two blits, using a separate intermediate surface.
Blitter		WaUse3dBlitForFBCMMIOProgramming	In this hang, BCS is executing LRI to an external GT address (address 50380, data 00000002). So instead of sending the cycle through the message channel, it will go through GAB. At the same time, BCS receives a GO=0 message from PM and this will bring the GO flag down which in turn will block the memory cycle from going out. The deadlock here is that the same state machine (in CSCFG) drives the GO ACK response and the external GT message.
Display	Clocks		CDCLK frequency change sequence needs added steps to prevent clock glitches. See North Display Engine Registers, CDCLK Sequence for details.
Display	Clocks		Very intermittent chance of failure of the display spread/bent clock reference. WA: Adjust the display spread/bent clock reference programming sequence when enabling/disabling HDMI, DVI, and DisplayPort.

Display	Clocks		The PCH display clock stops when BIOS enables PCH ISCLK PLL shutdown feature, causing backlight flicker and other problems. WA: When the ISCLK PLL shutdown feature is enabled, set a clock gate disable C2020h bit 12 before using Aux channel B/C/D, GMBUS, GTC, or panel power sequencing.

Display	Display		VGA may lockup and give a black screen randomly with some memory configs and/or memory tests. WA: See VGA_CONTROL for workaround bits since they change with project and stepping.
Display	Display		HDMI and DVI with audio are not supported when HSYNC Start is programmed equal to HBLANK Start.
Display	Display		FDI failures after training causes blank screen or hang. WA: Transcoder timing generator override bits must be set and cleared at certain point when enabling/disabling PCH transcoder. See BSpec Mode Set Sequences for exact workaround details.
Display	DisplayPort		There could be a rare case where the eDP modeset does not happen correctly if the link is already in "send normal pixels" when the pip e is enabled. WA: For eDP, do not set Normal Pixel at the end of Link Training. See DisplayPort Enable Sequence.
Display	DisplayPort		DP MST output incorrect for certain M and N and VC payload size values. WA: VC payload must be multiple of 4 in x1 lane config, 2 in x2, 1 in x4. See M/N Values.
Display	DisplayPort		DDIA Aux channel transactions get intermittent NAK errors with some receivers. WA: Increase DDI_AUX_CTL_A bits 27:26 Time out timer value to 600us 01b when doing DDIA aux transactions.
Display	DisplayPort		Big FIFO mode conflicts with enabling of DP Port Sync mode. WA: Set 0x45280 bits 2:1=11b before enabling port sync mode. See DisplayPort Enable Sequence.
Display	FDI		FDI Rx delay default value incorrect, causing FDI failures. WA: Set FDI delay to 90h before enabling FDI.
Display	FDI		FDI Rx TP1 time default value incorrect, causing FDI training failures.

			WA: Set TP1 to TP2 time to 48 clocks before enabling.
Display	FDI		Added programming needed to reset FDI mPHY IOSF-SB during mode set sequences. WA: See Mode Set Sequences for details.
Display	FDI		Added programming needed to control FDI Rx Pwrdn bits during CRT mode set sequences. WA: See Mode Set Sequences for details.
Display	FDI		Added programming needed to setup FDI mPHY registers at boot and in mode set sequences. WA: See Mode Set Sequences.
Display	General		TRANS_CONF Transcoder State may incorrectly show it is enabled before the transcoder has been enabled for the first time after reset or power well disable and enabled. This should not impact the normal usage of this field during the mode set disable sequence.
Power	IPS	WalpsWaitForPcodeOnDisable	It can take up to 42ms for pcode to complete IPS disabling. The driver sequence to disable IPS needs to account for this delay before disabling the last plane on the pipe, otherwise there will be corruption from disabling planes before IPS is completely disabled. See North Display Engine Registers, Intermediate Pixel Storage for details.
Power	IPS		Chance of screen corruption if IPS and display planes are not enabled and disabled in a specific sequence. WA: IPS cannot be enabled until at least one plane has been enabled for at least one vertical blank. IPS must be disabled while there is still at least one plane enabled. See North Display Engine Registers, Intermediate Pixel Storage.
Display	Panel fitter	WaPanelFitterDownscale	When using panel fitter downscaling (pipe source size is larger than panel fitter window size) the maximum supported pixel rate will be reduced by the downscale amount, and watermarks must be adjusted. Use panel fitter scale amount when calculating maximum pixel rate and watermarks.
Display	Panel power sequencing	WaVDDOverrideT4Power	When software clears the panel power sequencing VDD override bit from 1 to 0 (disable VDD override) it must ensure that T4 power cycle delay is met before setting the bit to 1 again, else panel ma y be damaged.

			WA: Use software timers to ensure T4 delay is met or use full panel power enable and not the VDD override.
Display	Power Off Sequence		On CDV when using Self Refresh mode and one pipe is active while the second pipe is being enabled or disabled, the workaround sequence below needs to be followed. This does not need to done when both pipes are being enabled or disabled at the same time. When turning off a second pipe: Disable second pipe using the power off sequence as normal. Anytime after the step in the disable sequence where there is a wait for pipe off status, re-enable Self Refresh if desired (register 0x20E0 bit 15 set to 1).
Display	Power On Sequence		On CDV when using Self Refresh mode and one pipe is active while the second pipe is being enabled or disabled, the workaround sequence below needs to be followed. This does not need to done when both pipes are being enabled or disabled at the same time. When turning on a second pipe: Disable Self Refresh (register 0x20E0 bit 15 set to 0). Wait for vblank from first pipe (frame is ended and the pipe will use only his part at the ddbm during next frames) Enable second pipe using the mode switch enable sequence as normal.
Power	PSR		PSR single frame update with sprite - Mask the sprite enabled when using single frame update with sprite. See North Display Engine Registers, SRD_CTL register for details.
Power	PSR		PSR single frame update - Mask register write events when using single frame update. See North Display Engine Registers SRD_CTL register for details.
KMD, Media		WaVeboxSliceEnable	Workaround required for checking Single Slice Enable flag in VEBOX State.
KMD, Media	Decode	WaJPEGHeightAlignYUV422H2YToNV12	Specific to JPEG decode. If the input is YUV422H_2Y and output is NV12 format, output picture height should be aligned by 16 bytes.
KMD, Media	Encode	WaAddMediaStateFlushCmd	Add Media State Flush command after Media Object Walker command.

			This WA is for a preemption hang, if the last command in the media batch buffer isn't a media-state-flush then a pre-emption on the following GPGPU batch buffer can cause a hang. A Media_State_flush command is required to be added at the end of media command buffer or the batch buffer with MEDIA_OBJECT/WALKER. Required for RCS engine in media pipeline.
KMD, Media	Encode	WaEnableVMEReferenceWindowCheck	WA for limiting the minimum downscaled surface dimensions to 48x48.
KMD, Media	Media (Decode)	WaInsertAVCFrameForFormatSwitchToJPEG	Insert a dummy AVC frame before a JPEG frame to WA format switch issue when Decode format is switched to JPEG.
Memory Views	Graphics Memory Address Type: GSM		Due to a workaround, first 4KB of DSM has to be reserved for GFX hardware use during render engine execution.
Memory Views	Memory Interface		32b PPGTT has no implications
Memory Views	Opaque Textures (DXT1_RGB)		The behavior of this format is not compliant with the OGL spec. As a workaround, the Surface Format should be set to BC1 and the Shader Channel Select A should be set to SCS_ONE.
Power	DPST		The DPST Histogram Event status may not clear if it is written with a 1b to clear it and the Histogram Interrupt enable field is changed from 0b to 1b in the same MMIO write. WA: To guarantee the event status is cleared, separate the single MMIO write into two writes.
Power	FBC		First line of FBC getting corrupted when FBC compressed frame buffer offset is programmed to zero. Command streamers are doing flush writes to base of stolen. WA: New restriction to program FBC compressed frame buffer offset to at least 4KB. See FBC_CFB_BASE in North Display Registers. Additionally the driver will need to avoid using this first 4KB of graphics data stolen memory for anything else.
Power	IPS	WaIpsDisableOnAsyncFlips	Enabling IPS increases the flip completion time results in lower frame rates or performance. WA: Disable IPS for Async flips.
Power	IPS		Programming requirement: IPS to be enabled only after display plane is enabled.

		WaIpsUpdateOnPlaneStatusChange	WA: IPS cannot be enabled until at least one plane has been enabled for at least one vertical blank. IPS must be disabled while there is still at least one plane enabled. See North Display Engine Registers Intermediate Pixel Storage.
Power	Turbo		Downward busyness interrupt not working when RC6 is enabled. Symptoms include missing interrupts making the feature generally unreliable. WA: SW should use another mechanism to decide when to requires a change in GT frequency. A suggestion would be to use PP1_C0_CORE_CLOCK as busyness counter. The register PUSHBUS_CONTROL - Push Bus Metric Control, MMIO: 0/2/0, Address 0A248h, bit 2, C0 Residency Time Enable controls this capability. It can be used to report GT activity that can then be used to determine how active GT has been over a given time period.
Power	RC6	WaRsUseTimeoutMode	EI counters are not reliable through RC6 entry / exit, and the resulting behavior effectively causes counters to lose time spent in RC6. WA: Use hardware RC6 Timeout Mode to replace hardware RC6 EI Mode to meet RC6 duty cycle targets. By setting the promotion time value to 800 microseconds, the RC timeout mode provides reasonable power / performance. The value may need to be tuned for specific workloads / systems. 1. Set RC_CTRL0 - Render C State Control 1, MMIO: 0/2/0, Address: 0A090h, bits 27:28 to 01b (time out mode). 2. Set RC_PROMO_TIME1 - RC Promotion Timer for RC6, MMIO: 0/2/0, Address: 0A0b8h to 0x271. (800 microsecond promotion timer value).
Power	PSR	WaMaskVBIduringSFU	PSR SFU not working during media playback. WA: Set PIPE_MISC MMIO 70030h bit20 to 1, masking VBI tracking during SFU enabled scenario.
Power	IPS	WaIpsDisableOnCdClockThreshold	WA:Disable IPS when dotclock > 95% of cdclk
Display	eDP	N/A	eDP link training failures if package C6 is entered when display on/off power well is disabled. The PCU thinks eDP is completely disabled and the display LCPLL can be shutoff, but eDP is actually partially enabled and needs the LCPLL. WA:Program SRD_CTL_EDP Max Sleep Time to 0 before starting eDP link training. This will tell the PCU to not turn off the LCPLL. Restore SRD_CTL_EDP Max Sleep Time after enabling TRANS_CONF_EDP.

Display	Dither	N/A	Dither pattern stops as soon as the transcoder configuration register is written to disable transcoder. Dither should continue until the vblank when transcoder actually disables. WA: A workaround is only needed if it is important to have a correct dither pattern in the final frame as display output is disabled. Any workaround must be specific to the implementation of the operating system and driver.
KMD		WaIdleLiteRestore	SW must always ensure ring buffer head pointer is not equal to tail pointer of a context, whenever it is submitted to HW for execution.
KMD	Mid-batch Preemption	WaDisableCtxRestoreArbitration	[Render CS Only][Runlist Mode of Scheduling]: SW must ensure arbitration is switched off while context restore is in progress for any given context. This is achieved by disabling arbitration by programming MI_ARB_ON_OFF to "Arbitration Disable" in RCS_INDIRECT_CTX buffer and by enabling back the arbitration by programming MI_ARB_ON_OFF to "Arbitration Enable" as the last command prior to MI_BATCH_END in the BB_PER_CTX_PTR buffer of every context submitted. Note that RCS_INDIRECT_CTX_OFFSET could be set to default value or any other legitimate value as per the programming notes of the register definition. • Arbitration disable by programming MI_ARB_ON_OFF (Arbitration Disabled) in RCS_INDIRECT_CTX buffer. • Arbitration enabled by programming MI_ARB_ON_OFF (Arbitration Enabled) as the last command prior to MI_BATCH_BUFFER_END in BB_PER_CTX_PTR buffer.
Blitter		WaUse3DBlitForNonCLAlignedLinearSrcDst	Color source and destinations must be restricted to CL alignment for linear addressing.

3D		WaGT3ResendDepthStencilAndHierDepthBufferState	Issue is BSPEC specifies X Max and Y Max are "exclusive" when they need to be "inclusive". Here is the BSPEC note for Clear Rectangle X Max in 3DSTATE_WM_HZ_OP: 'Specifies Xmax value of (exclusive) of clear rectangle with the Depth Buffer, used for clipping. Pixels with X coordinates greater than or equal to Xmax will be not be affected.'
Power	FBC		FBC may intermittently fail to update some lines after an image change, causing stale data to appear. The stale data can be prevented by manually invalidating FBC after flips and enabling FBC. Manual invalidation sequence: For (i=0 to 127) { Write 0x50344 = (i<<16) } The manual invalidation needs to happen in the frame that the image changes or FBC enables, between the start of vertical blank at the beginning of that frame and the start of vertical blank at the end of that frame. Recommended sequence: After an async flip, sync flip, or enabling FBC, wait for the next start of vblank (sync flips can alternatively wait for flip done), then perform the manual invalidation sequence.
KMD		WaSkipStolenMemoryFirstPage	WA to skip the first page of stolen memory due to sporadic HW write on *CS Idle.
KMD		WaForceContextSaveRestoreNonCoherent	To avoid a potential hang condition with TLB invalidation driver should enable masked bit 5 of MMIO 0x7300 at boot.
GMM		WaIommuCCInvalidationHang	When using IOMMU, IOTLB invalidation followed by a Context Cache invalidation can cause a hang. Workaround is to add a NOP (Device TLB invalidation is considered a NOP by HW) everytime before doing a context cache invalidation. This is done by writing dev_iotlb_inv_dsc to invalidation queue OR writing to scratch register for register based invalidation.(Scratch register[0x4F104] used for invalidation queue workaround(IommuUncoreUnavailable)) – Refer to BSpec for descriptor format - Search "dev_iotlb_inv_dsc".
GMM		Wa32bitGeneralStateOffset	Allocations of type Scratchflat/Instructionflat can only be referenced by 32 bit offset in a Graphics Segment. When Graphics memory size is >2GB, 32 bit offsets cannot address the 64 bit aperture segment. This workaround is used to define 32bit Segment to be used only for ScracthFlat/InstructionFlat heaps in >2GB configurations.

GMM		Wa32bitInstructionBaseOffset	Allocations of type Scratchflat/Instructionflat can only be referenced by 32 bit offset in a Graphics Segment. When Graphics memory size is >2GB, 32 bit offsets cannot address the 64 bit aperture segment. This workaround is used to define 32bit Segment to be used only for ScracthFlat/InstructionFlat heaps in >2GB configurations.
KMD		WaForceEnableNonCoherent	Must Force Non-Coherent whenever executing a 3D context. This is a workaround for a possible hang in the unlikely event a TLB invalidation occurs during a PSD flush. Set masked bit 4 in 0x7300 during boot.
KMD		WaDisableSRGBGammaToLinearConv	WA is to avoid Gamma to Linear conversion when source surface is SRGB and dest is non SRGB as per DX10's Present DDI expectation. In such a case, treat source as non-SRGB that will do bit-for-bit transfer from source to dest. E.g. if source format is B8G8R8A8_UNORM_SRGB and destination format is non-srgb then treat source as B8G8R8A8_UNORM instead by programming the surface state accordingly.
KMD		WaClearRenderResponseMasks	Need to set the correct render response masks to save power. HW power-on default is not good. Applies only to unmask primary plane flip and sprite flip done on A, B and C pipes. Done by clearing bit 1,2, 9, 10, 15 and 20 of MMIO 0x44050. Must be done at boot and all save/restore paths.
KMD		Wa4x4STCOptimizationDisable	HIZ/STC hang in hawx frames. W/A: Disable 4x4 RCPFE-STC optimization and therefore only send one valid 4x4 to STC on 4x4 interface. This will require setting bit 6 of reg. 0x7004. Must be done at boot and all save/restore paths.

KMD		WaModifyVFEStateAfterGPGPUPreemption	GPGPU preemption hang or corruption issue. This WA must be applied before re-submitting a GPGPU preempted workload. SW needs to do two things. 1) SW should detect if VFE unit in the context image have LRI commands to 0x54AC and 0x54B0. SW does that by checking if the LRI header at context image offset (0x0E90) <in VFE> is 0x11001003 and the two MMIO offsets in this LRI is 0x54AC and 0x54B0. If so, do this- D0 (LRI header): 0x1100_1003 -> 0x1100_ 1001 D1 (1st MMIO offset): 0x0000_54AC -> Leave this dword untouched D2 (1st MMIO offset value): 54AC_DATA -> Leave this dword untouched D3 (2nd MMIO offset): 0x0000_54B0 -> Overwrite to 0x0 D4 (2nd MMIO offset value): 54B0_DATA -> Overwrite to 0x0 2) Apart from it SW must also increment the Batch Buffer Address by Walker Command Length so that CS moves to the command following it. Size of Walker should be 0xF * sizeof(DWORD) = 0x3C Bytes.
KMD	RTL	WaBlockMsgChannelDuringGfxReset	When GPU hangs occurs, inside gfx driver TDR routine the driver writes to the reset register and after that it tries to read back the reset register and GPU never response back so system crashes. As a WA set 0x9424 bits 2,3,4,5,6,9 on boot.
KMD	RTL	WaFlushCoherentL3CacheLinesAtContextSwitch	Coherent L3 cache lines are not getting flushed during context switch which is causing issues like corruption. Need to set bit 21 of MMIO b118, then send PC with DC flush and then reset bit 21 of b118. This programming sequence needs to be part of the indirect context WA BB.
KMD		WaDisableSDEUnitClockGating	WA for GPGPU workload hang for which requirement is to disable SDE Unit clock gating. This is done by setting bit 14 of MMIO 9430.
KMD		WaIncreaseTagClockTimer	Clock gating causes wrong read address from State Array during RO invalidation causing Mem-Diff. So WA requires programing L3 tag clock timing register. At boot, write 0xB10Ch, bits [23:20] programmed to 1000b
KMD		WaDisableLiteRestore	There is a bug that can surface in Lite-Restore scenarios causing a hang. Driver needs to prevent lite restore scenarios from occurring. There are many ways to do this. 1) Driver can always set the 'Force Full Restore' bit in the context descriptor when writing to the submit port. 2) The driver can wait for the Active->Idle interrupt before submitting any new execution lists.
KMD		WaSetVfGuardbandPreemptionVertexCount	Workaround for potential 3d preemption bug that can cause data corruption. Driver should write to register 0x83A4 (preemption vertex count) and set a value of 0x20. At boot, write 0xffff0020 to 0x83a4 (it's a masked register).

KMD		WaProgramL3SqcReg1Default	Program the default initial value of the L3SqcReg1 for performance. This requires writing 0x784000 to MMIO b100.
KMD		WaSkipInvalidSubmitsFromOS	For Invalid submits from OS - simply report fence completion without submitting the DMA buffer to GPU.
KMD		WaTempDisableDOPClkGating	This WA is associated with another WA WaProgramL3SqcReg1Default (which requires doing a cpu write to 0xb100 offset with value 0x784000). Before applying WaProgramL3SqcReg1Default WA, SW should temporarily force disable render GOP clock gating by setting MMIO 0x9424 [bit 0] to 0. After WaProgramL3SqcReg1Default has been applied, S/W should restore 0x9424 with its old value.
3D		WaSendDummyVFEafterPipelineSelect	TSG unit writes null entries into context which can hang restore CS and TSG. WA: Send dummy VFE immediately after GPGPU pipeline select. We don't use Generic MediaStateClear at all.
3D			For indirect dispatch there is handle counter underflow and it leads to hang. WA: GPGPU walker need to use only curbe while doing indirect dispatch.
3D			Draw call doesn't send verticies to compute shader because 0-vertex draw call, preempted. Handful of reasons that VS would send no verticies. WA: Pipe control with CS stall only when the UAV is using a vertex shader.
3D		WaClearCCStatePriorPipelineSelect	Architecture hole; on GPGPU context restore, at the end of the context when CS sends a null prim, SVG and SARB does a state prefetch; by the time the data returns from memory, CS gates the FF clock. WA: In GPGPU mode, color cal state should not have valid bits. Before switching pipelines, send null CC state pointers.
3D		WaAvoidPMAStall	PMA bit is getting turned on when color traffic is in flight in RCPFE. WA: Put color flush before changing the PMA bit before stencil write cases.
3D			Increase the size of sub register from 8 to 16. For MOVI, did not change the code to accommodate. As a result, SW is not able to use this. WA: Use 2 SIMD16 instructions instead of 1 SIMD32. Will have slight performance hit.

3D			Write collision hang condition. When we have both remainder and quotient in same instruction, had write port collision on GRF, drop the dependency clear resulting in hang. WA: Break integer divide into separate quotient and remainder instructions.
3D			When DCS re-allocates handle ID, SDE flops but valids are not used in clock wakeup equations. When reallocation happens, handles lost. WA: Disable SDEunit clock gating. UCGCTL6[14]=1 (09430h[14])
3D			When we divide and denominator is DNORM, and DNORM is enabled, should produce infinity, but result is 0. Works OK if DNORMs are disabled, legacy features are fine. WA: Don't use DNORM on FDIV.
3D			GA is sending wrong data to fpu for src0 indirect addressing WA:When an instruction has a source region spanning two registers and a destination region contained in one register, one of the following must be true: The destination region is entirely contained in the lower Oword of a register. The destination region is entirely contained in the upper Oword of a register. The destination elements are evenly split between the two OWords of a register AND evenly split between the two source registers.
3D			MsaaBasic hang with single subspan dispatch enabled. WA: QtrCtrl must not be used for jmpi instruction.
3D			DF --> f format conversion for align16 has wrong emask calculation when the source is immediate. WA: In Align16 mode, format conversion from double-float to floats is not allowed when source is immediate data.
3D			SMOV instruction with byte data type. If this is used, corruption. Data read / write is not as expected. WA: Disable SMOV with byte.

3D		WaDisableObjectLevelPreemtionForVertexCount	Preemption the clock after the last vertex of draw call with less verticies than in the guardband, VF will indicate that it's incomplete, when it is actually complete. Will redraw the last instance. WA: For regular draw calls, when # od verticies is less than the GB, disable mid-object preemption.
3D		WaKVMNotificationOnConfigChange	DPR currently sends a "decreg" signal to DPCEUNIT for TRANS_CONF_EDP. The signal is expected only to pulse when a write on TRANS_CONF_EDP has occurred, but the signal is actually assigned to logic directly out of the automated register code which is read and write accesses.
3D		WaForceDX10BorderColorFor64BPTTextures	Dgunit sending wrong border color data for DX9 and 64 bpt. WA: Force xmajor for 64bpt non-aniso in DX9 mode.